Berliner Informatik-Tage

bit '85

Humboldt-Universität zu Berlin

Tagungsbericht Band 3

SEKTION 5
Sprachaspekte und künstliche Intelligenz

Klaus Ahrens

" SIMULATIONSWERKZEUGE IN MODULA-2 "

Die Anwendung der computergestützten Simulation entwickelt sich immer mehr zu einer notwendigen Voraussetzung im Entwurfsprozeß komplexer Systeme. Dabei spielen in den letzten Jahren in verstärktem Maße Rechnersysteme selbst als Untersuchungs- und Entwurfsobjekte eine wichtige Rolle. Zu den Faktoren, die diese Tendenz beschleunigen, gehören vor allem folgende:

(1) Die explosive Entwicklung des Hardwaresektors ist Ausgangspunkt einer weitreichenden Anwendung von Computern zur Steuerung realer Prozesse. In den Entwurf derartiger Steuerungssysteme sind sowohl Hardware als auch Software integriert. Der Entwurf geht also nicht mehr nur von einer festen Hardware aus und untersucht alternative Softwarelösungen, sondern bezieht die Suche nach optimalen Hardwarekonfigurationen mit ein. Damit ergibt sich für die Simulation die Forderung nach solchen Simulationssystemen, die diesen Aspekt mit berücksichtigen, indem sie Experimente mit unterschiedlichen Software-, Hardware- und Lastmodellen ermöglichen.

(2) Im Zusammenhang mit dem ersten Faktor steht die Forderung nach einem notwendigen Vorlauf in der Softwareentwicklung. Mit dem Entwurf der Programme für neue Rechnerkonfigurationen kann nicht erst begonnen werden, wenn diese verfügbar sind. Im Gegenteil muß die Softwareerstellung unabhängig von Fragen der Verfügbarkeit der Zielhardware unterstützt werden, so daß eine weitgehend ausgetestete Software und die entsprechende Hardware möglichst gleichzeitig verfügbar werden. In Cross-Entwicklungssystemen dieser Art wird die Simulation insbesondere dann unumgänglich, wenn es über eine bloße Emulation der Zielmaschine hinaus um eine Einbeziehung von Modellen realer Prozesse bzw. um die Ableitung quantitativer Aussagen

über das dynamische Systemverhalten geht.

(3) Die Forderung nach zuverlässigen Steuerungssystemen, die überdies auch eine leichte Wartung und Portabilität unterstützen, führt zwangsläufig dazu, daß auch in der Systemprogrammierung die traditionellen Assemblersprachen zunehmend von modernen höheren Programmiersprachen abgelöst werden. Dabei erhebt sich jedoch sofort die Frage nach resultierenden Uneffektivitäten. Die Simulation muß Mittel bereitstellen, die die Engpässe derartiger Systeme in höheren Programmiersprachen lokalisierbar machen, so daß in jedem konkreten Fall entschieden werden kann, ob und welche Teile des Gesamtsystems maschinennah zu implementieren sind, um den Anforderungen (insbesondere Zeitanforderungen bei Realzeit-Anwendungen) gerecht zu werden.

Existierende Simulationssysteme werden diesen neuen Anforderungen nur in unzureichendem Maße gerecht (GPSS- Familie). Auch wenn sie zum Teil die Modellerstellung und -untersuchung von Hardware- und Softwarekomponenten unterstützen, ermöglichen sie für uns keinen direkten Übergang von Softwaremodellen zur eigentlichen Implementation der Software.(Simula-OASIS /1/) Die leichte Überführbarkeit von realen Implementationen von Programmen zu entsprechenden Modellen und umgekehrt ist jedoch eine entscheidende Voraussetzung für einen integrierten Implementations- und Simulationsprozeß im Entwurf moderner Steuerungssoftware, die insbesondere dann gegeben ist, wenn beide Phasen in <u>einer</u> Sprache erfolgen.

Ausgehend von dieser Analyse wurde in der Forschungsgruppe "Simulation" am Bereich Informationsverarbeitung der Humboldt-Universität Berlin untersucht, inwieweit die Sprache Modula-2 /2/ ein geeigneter Ausgangspunkt für einen solchen integrierten Entwurfsprozeß ist. Diese Sprache wurde aus folgenden Gründen gewählt:

- Modula-2 ist ursprünglich für die Systemprogrammierung auf Kleinrechnern entwickelt worden, erfüllt also den Implementationsaspekt a priori;

- Modula-2 enthält ein elementares Koroutinenkonzept, welches als Ausgangspunkt für die quasiparallele Modellierung paralleler Prozesse als geeignet erschien;
- Die Sprache Modula-2 hat seit ihrer ersten Implementation an der ETH Zürich eine rasche Verbreitung und Anwendung gefunden, was sich sicherlich in Zukunft noch fortsetzen wird;
- Modula-2 ist eine Pascal-ähnliche Sprache mit einer systematischen Syntax, einem modernen Typkonzept und mit der Möglichkeit der separaten Übersetzung, wobei die Konsistenz der Schnittstellen durch Compiler überprüfbar ist; Somit wird sie dem Pascal-Programmierer leicht zugänglich, bietet zugleich aber eine geeignete Ausgangsbasis auf einem möglichen Weg zu Ada;
- Modula-2 ist am Bereich Informationsverarbeitung seit 1982 auf K1620/1630 und seit kurzem auch auf Bürocomputer verfügbar.

Der erste Ansatz bei der Untersuchung von Modula-2 für die Simulation bestand in der Implementation eines allgemeinen Basissimulationssystems in dieser Sprache. Dazu bot sich als Vorbild die Simula-Klasse 'Simulation' /3/ an, da sie

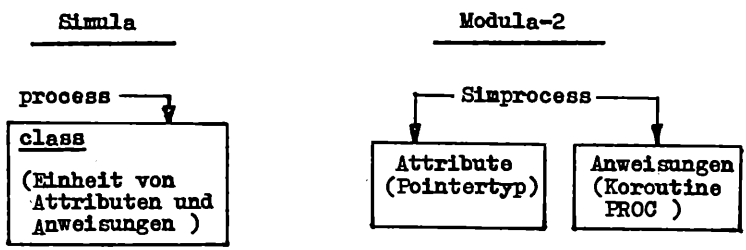
1. typische Konzepte eines prozeßorientierten Basissimulationssystems in sich vereinigt, die auch Ausgangspunkt für Simulationssysteme in anderen Sprachen waren und sind (Pascal, Ada,..);
2. als Basis für verschiedene Erweiterungen (Unterklassen) zu fachspezifischen Simulationssystemen verwendet wurde, die zum Teil auch in unserer Forschungsgruppe entwickelt wurden, so daß hier bereits einige fertige Systeme, aber auch wesentliche Erfahrungen in der Softwaretechnologie für Simulationssysteme allgemein vorlagen.

Um ein solches 'Simulation'-Konzept auch in Modula-2 zu realisieren, mußten zunächst die wesentlichen Unterschiede beider Sprachen hinsichtlich dieser Zielstellung erfaßt werden.

	Modula-2	Simula
dynamische Strukturen	Pointertypen	Klassen
abstrakter Datentyp	Moduln	Klassen
Koroutinen	Prozeduren	Klassen
Ein- und Ausgabe	- (Streams, InOut)	BASICIO
Textverarbeitung	Strings	text
Zufallszahlengeneratoren	-	Standard
hierarchische Erweiterungen	- (EXPORT / IMPORT)	Unterklassenkonzept

(dabei bedeutet '-', daß entsprechende Konzepte fehlen, bzw. nur bedingt, als zu erweiternde Basiselemente vorhanden sind)

Es zeigt sich, daß der universelle Charakter der Simula-Klassen kein adäquates Modula-2 -Gegenstück hat, so daß einige Grundkonzepte neu zu überdenken waren. Die folgende Übersicht gibt lediglich eine Skizze der allgemeinen Vorgehensweise, die bei der Implementation verfolgt wurde. Auf eine eingehende Dokumentation der Erstellung des Moduls 'SIMULATION' soll hier verzichtet werden (entsprechende Ausführungen sind in /4/ zu finden).



Für die Modula-2 Umsetzung eines Prozeßbegriffs der diskreten Simulation macht sich also eine Aufspaltung in beschreibende Daten (Pointertyp auf ein Record, welches die Attribute eines Prozesses, sowie ein besonderes Feld des Typs PROCESS enthält) und in den Anweisungsteil erforderlich. Dadurch ist es möglich, die Dynamik von Prozessen auch in Modula-2 aufrecht zu erhalten und zugleich die Koroutinen (PROCESS) zu verwenden. Die Verbindung beider Komponenten eines Prozesses (Simprocess) erfolgt bei seiner Erzeugung durch die Prozedur 'NewSIMPROCESS'.

Im Ergebnis des ersten Ansatzes der Nutzung von Modula-2 für die diskrete Simulation entstand der Modul 'SIMULATION', dessen Definitionsmodul hier in Auszügen folgt.

```
DEFINITION MODULE  Simulation;
EXPORT QUALIFIED ...(* alle definierten Objekte *)...;

TYPE   Simprocess;  (* hidden *)
VAR    Main: Simprocess;

PROCEDURE  Hold (T: REAL); (* verzögert den aufrufenden
                              Simprocess um die Zeit 'T' *)
PROCEDURE  Passivate;  (* verzögert den aufrufenden Simpro-
                          zess für unbestimmte Zeit (er wird
                          passiv)  *)
PROCEDURE  Current (): Simprocess; (* liefert den aktuellen
                                      Simprocess  *)
PROCEDURE  Time (): REAL;  (* berechnet die aktuelle Simu-
                              lationszeit *)

  ....
PROCEDURE  Activate (SIMP: Simprocess); (* aktiviert SIMP *)
PROCEDURE  ActivateAT (SIMP: Simprocess; T: REAL;
                       PRIOR: BOOLEAN); (* aktiviert SIMP
                          zum Zeitpunkt 'T' mit der Priorität
                          PRIOR   *)

  ...
PROCEDURE  NewSIMPROCESS (BODY: PROC; n: CARDINAL):Simprocess;
           (* erzeugt einen neuen Simprocess mit den
              Anweisungsteil BODY und mit einem Arbeits-
              speicher der Größe 'n'  *)

  ...

END  Simulation.
```

Die zweite Stufe der Untersuchungen bestand in der Implementation eines fachgebietsorientierten Simulationssystems in Modula-2. Entsprechend der globalen Zielstellung der simulativen Unterstützung des Entwurfsprozesses von Rechnersystemen wurde dabei ein solches Syatem erstellt, welches spezielle Bausteine für die Modellierung in diesen Bereich bereitstellt. Auch hier kam die Anregung aus der Sprache Simula, in der uns das System OASIS vorlag, das für diesen Zweck geeignet erschien. Da OASIS selbst eine Unterklasse von 'Simulation' ist, war der Versuch, ein analoges System in Modula-2 zu implementieren zugleich eine Anwendung des Moduls 'SIMULATION' in der sich die Tragfähigkeit der entwickelten Konzepte erweisen konnte. Das System OASIS besteht in seiner Simula-Implementation im wesentlichen aus einer Sammlung von elementaren Bausteinen, so daß es sich für eine Dekomposition in Modula-2 unter Nutzung separater Moduln eignet. Das gesamte Simulationssystem ist von relativ geringem Umfang. Seine Konzeption liegt gerade in der Bereitstellung einfacher, aber erweiterungsfähiger Grundelemente, die typische Komponenten von realen Rechnersystemen modellmäßig repräsentieren.

Unter Nutzung des Moduls 'SIMULATION' wurde das System OASIS-M2 implementiert, das sich aus folgenden Bestandteilen (separate Definitions-/Implementationsmoduln) zusammensetzt:

- MODULE frame	: Experimentinitialisierung und -abschluß;
- MODULE reports	: Routinen für die Datensammlung -aufbereitung und -ausgabe;
- MODULE devices	: Basistyp für Geräte mit entsprechenden Routinen;
- MODULE memoryhandling	: Basistypen zur Modellierung von Speichermedien;
- MODULE modulehandling	: Basistyp für Prozessoren und Programme mit entsprechenden Beziehungen.

Auf dem Weg zu einem integrierten Entwurfsprozess für Modula-2 -Steuerungssoftware ist das hier vorgestellte Simulationssystem OASIS-M2 sicher nur ein erster Schritt. Weitere Untersuchungen müssen vor allem in zwei Richtungen

weiterverfolgt werden:

(1) Modifikation und Weiterentwicklung des OASIS-M2-Ansatzes, um die Spezifik der Sprache Modula-2 als Systemprogrammierungssprache besser zu berücksichtigen und auszunutzen;

(2) Verfolgung weiterer möglicher Ansätze der Simulation, die nicht auf next-event-scheduling Algorithmen beruhen und somit Bereitstellung eines ganzen Systems von Simulationswerkzeugen in einer Modula-2-Programmierungebung.

Literatur

/1/ Unger, B.W.; Bidulock, D.S.; Parker, J.R.: OASIS 3.0 Reference Manual. Departement of Computer Science, University of Calgary, Research Report No. 81/58/10, March 1981.

/2/ Wirth, N.: Programming in Modula-2. Springer-Verlag Berlin/Heidelberg/New York, 1982.

/3/ Dahl, O.J.; Myhrhaug, B.; Nygaard, K.: Simula 67 - Common Base Language. Norwegian Computing Centre, Oslo 1968.

/4/ Ahrens, K.: Modula-2 als Basissprache für die Modellierung und Simulation paralleler Prozesse. in Vorbereitung für die 'Seminarbericht'-Reihe der Humboldt-Universität Berlin 1985.

Anschrift des Autors:

Klaus Ahrens, Bereich Informationsverarbeitung
Sektion Mathematik der Humboldt-Universität
DDR - 1086 B e r l i n

PSF 1297

Clauß, Matthias
Fischer, Günther

Der ESER-C-Compiler unter PSU

Systemsoftware unterliegt extremen Anforderungen hinsichtlich Kodeeffizienz, Zuverlässigkeit, Wartungsfreundlichkeit sowie in zunehmendem Maße auch hinsichtlich der Übertragbarkeit auf andere Rechnerarchitekturen.
Moderne höhere Programmiersprachen unterstützen die Entwicklung gut strukturierter, zuverlässiger und portabler Programme. Trotzdem wird insbesondere in der Systemprogrammierung auch noch gegenwärtig vor allem auf Grund geringeren Speicherplatzbedarfes und günstigeren Laufzeitverhaltens oft auf Assembler zurückgegriffen, obwohl damit erhebliche Nachteile, wie verlängerte Programmentwicklungszeiten usw. verbunden sind.

Die Programmiersprache C wurde 1972 von Dennis M. Ritchie entworfen und implementiert. Sie bietet eine leistungsfähige Schnittstelle zur Hardware und stellt geeignete Mittel zur Entwicklung portabler und kodeeffizienter Programme zur Verfügung. Ein Grundprinzip von C besteht darin, dem Programmierer bei der Wahl der sprachlichen Mittel möglichst wenig Restriktionen aufzuerlegen. Die damit verbundene Flexibilität kommt vor allem bei der Implementation von Systemsoftware zum Tragen. Andererseits wird dadurch vom Programmierer ein hohes Maß an Disziplin und Sorgfalt gefordert, insbesondere bei der Verwendung externer Variabler und im Umgang mit Zeigern.

Im Zusammenhang mit dem Betriebssystem UNIX findet die Sprache C international wachsende Verbreitung. C-Compiler existieren mittlerweile für die unterschiedlichsten Rechnerarchitekturen über die gesamte Breite von Mikrorechnersystemen bis zu Hochleistungsrechnern.

Die Entwicklungsarbeiten an einem C-Compiler für ESER-Anlagen werden seit Mitte 1984 auf Grundlage einer Kooperationsvereinbarung zwischen der TH Karl-Marx-Stadt und dem LfA Berlin weitergeführt. Ab I/85 ist der Compiler zur Erprobung freigegeben und bisher an 8 Hochschuleinrichtungen installiert. Außerhalb des MHF kann eine Vorabnutzung mit dem LfA Berlin vereinbart werden.

Die Hauptbestandteile des C-Compilers sind der Präprozessor, 2 Übersetzungspässe, Optimierungspaß und die Standard-E/A-Bibliothek, die auch die Schnittstelle zum Filemanager der PSU /1/ bereitstellt. Der realisierte Sprachumfang entspricht der C-Sprachbeschreibung von Kernighan und Ritchie /2/. Darüberhinaus wurden Strukturzuweisungen und der Aufzählungstyp in das Sprachkonzept aufgenommen. Die C-Compiler für 16-Bit-Systeme in der DDR (K1630, SKR, U8000, ACS) haben bzw. werden den gleichen Sprachumfang realisieren. Hierzu finden Abstimmungen zwischen den Entwicklern statt, um den Softwareaustausch zu erleichtern.

Entwicklungsprobleme

Für den ESER-C-Compiler sind auf Grund des stark abweichenden Maschinenbefehlssatzes einige Besonderheiten zu beachten.

- Einzelfunktionen können durch Basisregisteradressierung bei Verwendung eines Registers nur 4K groß sein. Diese Begrenzung ist selten problematisch. Falls erforderlich, kann der Programmierer zusätzliche Basisregister vereinbaren. Die Anzahl der möglichen Registervariablen wird dadurch reduziert.

- Da in C häufig mit externen Variablen gearbeitet wird, kann der Programmierer solche Variablen mit dem Schlüsselwort common direkt adressierbar machen.
  Mit Hilfe der Präprozessorsteuerung sind in beiden Fällen Portabilitätsprobleme vollständig vermeidbar.

- Die Mehrfachdefinition nichtinitialisierter Variabler ist in C nicht verboten. Das Syntaxprüfprogramm lint /3/, das unter anderem Portabilitätsprobleme aufspürt, erkennt einen solchen "saloppen" Programmierstil.
  Bei SKR-Maschinen erfolgt die Speicherplatzvergabe nach initialisierten (.DATA) bzw. nichtinitialisierten Variablen (.BSS). Der Programmverbinder realisiert die gewünschte Zuordnung und der Lader initialisiert die verbleibenden Bereiche zu Null, wie es vom Sprachstandard gefordert wird. Die bisher verwendeten OS-Komponenten Assembler, Verbinder und Lader lassen eine solche Vorgehensweise nur mit Kompromissen zu.
  Endgültig und effektiv sind diese Probleme nur durch einen speziellen Assembler, Programmverbinder und Lader lösbar.

Codeoptimierungen

Ein Ziel der C-Programmierung ist es, möglichst effektive Maschinenprogramme zu erhalten. Darauf wird an verschiedenen Stellen Einfluß genommen.

- Der Programmierer kann bereits auf Quelltextebene das Codeverhalten beeinflussen. Beim ESER-C-Compiler kann man bis zu 6 Variable pro Funktion direkt im Register halten. Insbesondere bei Zeiger auf Strukturen und Felder sind dabei Codeeffektivierungen zu erwarten. Registervariable bringen im Vergleich zu automatischen Variablen auch keine zusätzliche Stackbelastung, was sich insbesondere bei stark rekursiven Funktionen vorteilhaft auswirkt.
  Jeder Ausdruck besitzt einen Wert. Außerdem können Wertzuweisungen wie gewöhnliche Ausdrücke verwendet werden. Die sich daraus ergebenden Möglichkeiten zur kompakteren Schreibweise unterstützen den Prozeß der Codeoptimierung.
- Die Codegenerierung basiert auf dem Sethi-Ullmann-Algorithmus /4/. Dabei erfolgt eine Vielzahl von Optimierungen auf dem Ausdrucksniveau. Die Ausdrücke, die in Form von Binärbäumen vorliegen, werden untersucht und erforderlichenfalls umgestellt und vereinfacht. Die Fähigkeiten der Maschine zur Adreßbildung auf der Basis von Registerinhalten und Offsets werden dabei weitestgehend ausgenutzt.
- Wahlweise ist eine Optimierung auf Assemblerniveau (peephole-Optimierung /5/) möglich.
  Dabei werden z.B. Sprünge zu Sprüngen und überflüssige Ladebefehle beseitigt sowie Adressierungen vereinfacht.
  Zur Optimierung wird eine vollständige Funktion herangezogen.
  Die bisherigen Tests führten zu Codeverbesserungen von 5-10%.

Erfahrungen beim Softwaretransport

Im Verlauf des vergangenen Jahres wurden an der MLU Halle, der KMU und der TH Leipzig sowie der TH Karl-Marx-Stadt ca. 40 Programme aus dem Betriebssystem MUTOS 1630 /6/ in die PSU transportiert. Weiterhin realisierte das LfA Berlin die Entwicklung des Optimierungspasses für ESER im System MUTOS 1630. Der anschließende Transport in die PSU erwies

sich als problemlos. Zu den implementierten Komponenten gehören auch solche komplexen Werkzeuge wie nroff, yacc, lex, sort, m4, sed und die grep-Familie.
Der Übertragungsaufwand kann wie folgt klassifiziert werden:

- Programme, die mit der Absicht des Transportes auf andere Maschinen geschrieben wurden (z.B. yacc und lex), braucht man nur neu zu übersetzen. Der lint ist ein Werkzeug zur Unterstützung dieser Absicht.

- Manche Programme nehmen direkt auf die Maschinenarchitektur Bezug (z.B. integrale Grenzen, Adreßgröße). Solche Probleme müssen "nur" erkannt werden und sind dann leicht korrigierbar. Mitunter sind solche Abhängigkeiten in define-Anweisungen des Präprozessors enthalten.

- Bei einer weiteren Gruppe von Programmen ist der Algorithmus an den ASCII-Zeichensatz gebunden. So sind z.B. bei nroff und sed im freien 8. Bit Steuerinformationen untergebracht. In solchen Fällen ist es meist erforderlich, die Algorithmen sorgfältiger zu analysieren.
  Eine Lösungsmöglichkeit dieses Problems besteht darin, programmintern im ASCII-Zeichensatz zu arbeiten. Dieser Weg wurde beim nroff gewählt. Zu beachten ist dabei die notwendige Umstellung von Zeichenkonstanten in eine binäre ASCII-Darstellung.
  Bei komplexen Algorithmen erscheint dieser Aufwand vertretbar. Eine alternative Möglichkeit stellt die Änderung der Programmlogik dar.
  Eine Abschätzung des für die Lösung derartiger Probleme notwendigen Aufwandes kann nicht allgemein angegeben werden.

Weitere Arbeiten

Ende 1985 wird der ESER-C-Compiler unter PSU einschließlich der Komponenten lint und make /3/ zur produktiven Nutzung bereitgestellt. Der Vertrieb erfolgt durch den VEB Lfa Berlin. Darüberhinaus laufen Arbeiten zur Entwicklung eines in C geschriebenen und in die PSU integrierten Assemblers sowie Programmverbinders. Um die Softwareentwicklung auch in der Phase der interaktiven Programmtestung zu unterstützen, sind in

einer weiteren Etappe Entwicklungsarbeiten zur Bereitstellung einer zu adb /3/ vergleichbaren Komponente vorgesehen.

Ein wichtiger Aspekt besteht in der Untersuchung von Spracherweiterungen. In der Literatur werden hierzu verschiedene Arbeiten vorgestellt, so z.B. die Realisierung des Class- bzw. des Modulkonzeptes /7/, /8/ und die Bildung abstrakter Datentypen /9/. Andere Konzepte werden in /10/ und /11/ diskutiert. Die Spracherweiterungen werden i.a. durch Präprozessoren realisiert und stehen damit dem Programmierer wahlweise als zusätzliche Werkzeuge bei der Softwareentwicklung zur Verfügung.

Literatur

/1/ PSU-Systemhandbuch, Programmschnittstelle, VEB LfA Berlin, 1984, S032-02-001

/2/ B. W. Kernighan and D. M. Ritchie, The C Programming Language, Prentice Hall, 1978

/3/ UNIX - Programmer's Manual. Seventh Edition, Vol1, 1979, Bell Telephon Laboratories

/4/ R. Sethi, J. D. Ullmann, The Generation of Optimal Code for Arithmetic Expressions, in B. W. Pollack, Compiler Techniques, AUERBACH Computer Science Series, 1972, 229-247

/5/ W. M. McKeeman, Peephole Optimization, in B. W. Pollack, Compiler Techniques, AUERBACH Computer Science Series, 1972, 225-228

/6/ MUTOS - Anwenderdokumentation, Robotron, 1985

/7/ B. Stroustrup, Adding Classes to the C Language: An Exercise in Language Evolution, Software Practice and Experience, 13, 1983, 139-161

/8/ S. Boyd, Modular C, SIGPLAN Notices, 18(4), 1983

/9/ S. Boyd, Free and Bound Generics: two techniques for abstract data types in Modular C, SIGPLAN Notices, 19(3), 1984

/10/ J. Katzenelson, Introduction to Enhanced C, Software Practice and Experience, 13(8), 1983

/11/ T. A. Budd, An Implementation of Generators in C, Computer Language, Vol 7, 1982, pp 69-87

Gellermann, W. Dietmar
Grabowski, Jan

Konzept für einen Silicon-Compiler

1. Silicon-Compiler

Gegenwärtig wird weltweit an durchgängig rechnergestützten Verfahren zum Entwurf hoch- und höchstintegrierter Schaltkreise gearbeitet. Dabei spielen u. a. folgende Kriterien eine Rolle:

(1) Resultat des Verfahrens ist eine vollständige, verifizierte Layoutbeschreibung.

(2) Das Verfahren wird vom Nutzer über eine einheitliche Nutzerschnittstelle gesteuert, bestehend etwa aus einer Steuersprache, strukturellen und funktionellen Beschreibungsmitteln, einer Datenbasis von Standardbausteinen und deren Eigenschaften sowie aus entsprechenden Nachrichten, die auf dem Niveau dieser Schnittstelle interpretiert werden können, also nicht auf abgeleitete Datenstrukturen bezug nehmen.

Ein Entwurfsverfahren, das diesen beiden Kriterien genügt, wird heute verschiedentlich als "Silicon-Compilation" bezeichnet. (vgl. /1/) Die Analogie zum Compiler einer höheren Programmiersprache ergibt sich vor allem aus der Bedingung (2), die den Benutzer vom Umgang mit den im Entwurfsprozeß erzeugten Zwischen- und Endresultaten befreit (vgl. /2/). Eine strengere Analogie zum Compiler würde sich ergeben, wenn Bedingung (2) verschärft wird zu:

(3) Die Nutzereingabe besteht im wesentlichen aus einer Beschreibung des von der Schaltung zu realisierenden Algorithmus sowie wenigen, auch vom naiven Nutzer in ihren Wirkungen leicht überschaubaren "Pragmas" (Steuerungsoptionen, die die Semantik nicht beeinflussen).

Ein Silicon-Compiler mit den Eigenschaften (1) - (3) ist potentiell geeignet für den nichtprofessionellen Anwender; damit kann das Entwerfen von Schaltkreisen eine vergleichsweise leichte Aufgabe werden, ähnlich wie der Einsatz von höheren Programmiersprachen das Programmieren zu einer leichten Aufgabe gemacht hat. Voraussetzung ist hier wie dort, daß der Preis, der in Form unvermeidlicher Effizienzverluste gezahlt wird, als nicht zu hoch angesehen wird.

2. Höhere Beschreibungssprachen für integrierte Schaltkreise

Bedingung (3) weist der Hardwarebeschreibungssprache (HDL) einen wesentlichen Platz im Entwurfsinstrumentarium ein. Es gibt schon eine ansehliche Zahl von Hardwarebeschreibungssprachen; in jüngster Zeit geht das Bestreben dahin, VLSI-fähige HDL's zu entwickeln /3/. Einige dieser Sprachen unterscheiden sich nur punktuell von höheren Programmiersprachen, z. B. CHISEL /4/, MIMOLA /5/. Ein notwendiger Unterschied besteht dort, wo

- Konstruktionen weggelassen werden, die sich nicht in Hardware übersetzen lassen (dynamische Variable!),
- die Ausdrucksmöglichkeiten für Parallelität in angemessener Weise erweitert werden,
- Pragmas zur Steuerung des Entwurfsprozesses verwendet werden.

Grundsätzlich erscheint es sinnvoll, insbesondere die Konzepte von höheren Programmiersprachen auf HDL's zu übertragen, die den objektorientierten und modularen Entwurf unterstützen. Ein schönes Beispiel einer solchen HDL ist ZEUS /6/.

3. Die Sprache TDSL

An der Humboldt-Universität wurde ein Konzept für eine Hardwarebeschreibungssprache TDSL (top-down silicon language) vorgestellt /7/. Dabei wurden folgende Merkmale angestrebt:

- Beschreibungsmittel für parallele Signalflüsse,
- Prozedur- und Datenabstraktion,
- Modularität,
- parameterabhängige Typen,
- rekursive Übersetzung,
- Lokalität der Datenübergabe.

Eine einfache Anweisung von TDSL hat die Form

$$c \rightarrow y := x \rightarrow d$$

Hierin sind y und x gewöhnliche Variable, und c und d sind Variable des Typs <u>control</u>. Letztere sind verantwortlich für die Ablaufsteuerung; sie verhalten sich wie die Plätze in einem Petri-Netz, d. h. jede control-Variable kann nur den Wert <u>true</u> annehmen; bei Abarbeitung der obigen Anweisung verliert c ihren Wert, und d bekommt einen Wert.

Anweisungen, die durch Semikolon getrennt sind, sind zur parallelen Ab-

arbeitung bestimmt; benutzen sie gemeinsame control-Variablen, so können Konflikte entstehen, die dadurch gelöst werden, daß eine der Anweisungen nicht aktiviert wird; in

```
b1, c→y := x→d;
b2, c→y := x→e;
```

kann z. B. stets nur eine der beiden Anweisungen ausgeführt werden.
An Stelle einer Menge von Anweisungen kann ein Prozeduraufruf geschrieben werden, der auf eine an anderer Stelle gegebene Prozedurdefinition Bezug nimmt; in Prozedurdefinitionen verwendete Variable sind entweder lokale Variable oder in-Parameter oder out-Parameter; zwei ausgezeichnete Variable entry und return von Typ control sind als in- bzw. out-Parameter stets implizit vorhanden. Wird z. B. die Prozedur

```
procedure p(in x, out y: boolean)
```

durch eine Anweisung

```
entry→ A→return
```

definiert, so ist ein Aufruf

```
c→p(x,y)→d
```

äquivalent mit den drei Anweisungen

```
c→entry;
entry→A→return;
return→d;
```

Ein Modul ist eine verallgemeinerte Variable, die außer Variablen auch Prozeduren als Bestandteile enthält. Ein Modul wird in einer Typdefinition erklärt. Beispiel einer Typdefinition:

```
type box;
     procedure set;
     procedure res;
     procedure test(out t: boolean);
privat
     b: boolean;
     procedure set;
          entry→b := true→return;
     end set;
     procedure res;
          entry→b := false→return;
     end res;
     procedure test(t);
          entry→t := b→return;
     end test;
end box
```

Ein Modul wird erzeugt, indem ein Objekt des Modultyps vereinbart wird; die Vereinbarung

```
bo: box
```

erzeugt einen Modul des Typs box und ermöglicht z. B. einen Prozeduraufruf

```
bo.set
```

Im Unterschied zu Programmiersprachen ist für TDSL wesentlich, daß man auf diese Weise mehrere Moduln eines Typs erzeugen kann, die dann vom Compiler in mehrere physische Einheiten transformiert werden. Jede Prozedur ist genau so oft physisch verfügbar, wie der Modul erzeugt wurde, in dessen Typdefinition sie steht.

Konstanten können in TDSL als Typparameter verwendet werden; so bezeichnet

```
type register(const N: integer) = array[Ø .. N - 1] of boolean
```

einen parameterabhängigen Typ. Zur Übersetzungszeit müssen für alle Konstanten Werte eingesetzt bzw. berechnet sein.

Laufanweisungen werden zur Übersetzungszeit ausgewertet, z. B. bedeutet

```
for i := Ø to N - 1 do y[i] := x[i] od
```

daß ein N-bit-Register-Transfer stattfindet.

Rekursive Übersetzung kommt zustande, wenn die Definition eines (parameterabhängigen) Typs sich selbst aufruft. Das folgende Beispiel zeigt einen rekursiv zu übersetzenden Typ.

4. Beispiel: Der rekursive Multiplizierer

Der rekursive Multiplizierer multiplier(N) für zwei N-bit-Zahlen, wobei N eine Zweierpotenz ist, enthält vier Multiplizierer multiplier(N div 2), zwei Addierer, vier connect- und zwei shift-Moduln. (connect und shift werden nachstehend definiert. Der Addierer wird als bekannt vorausgesetzt).

```
type bool(const N: integer) = array[ø .. N - 1] of boolean;

type adder(const N: integer);
   procedure ad(in x, y: bool(N), out z:bool(N));
private
   . . .
end adder;

type connect(const N: integer; const high: boolean);
   procedure co(in r: bool(N); out s: bool(N div 2));
private      (* es folgt der nichtexportierte Teil von connect *)
   procedure co(r,s);
        entry→
             for i := ø to N div 2 - 1 do
                  assume high;   (* Fallunterscheidung zur Übersetzungs-
                  s[i] := r[i + N div 2];                    zeit *)
                  assume not high;
                  s[i] := r[i];
                  endassume
             od
        →return
   end co
end connect;

type shift(const N: integer);
   procedure sh(in r: bool(N); out s: bool(N * 2));
private
   procedure sh(r,s);
        entry→
             begin
                  for i := ø to N div 2 - 1 do s[i] := false od;
                  for i := N div 2 to N + N div 2 - 1 do
                       s[i] := r[i - N div 2] od;
                  for i := N + N div 2 to 2 * N - 1 do s[i] := false od;
             end
        →return
   end sh
end shift;
```

```
type multiplier(const N: integer);
   procedure mu(in x, y: bool(N); out z: bool(N * 2));
private
   assume N > 1;
        add1, add2: adder(N * 2);
        mult: array[1 .. 2, 1 .. 2] of multiplier(N div 2);
        shift1, shift2: shift(N);
        conn1, conn2, conn3, conn4: connect(N, true);
        conn5, conn6, conn7, conn8: connect(N, false);
        u, v: array[1 .. 2, 1 .. 2] of bool(N div 2);
        w: array[1 .. 2, 1 .. 2] of bool(N);
        r, s1, s2, t: bool(N * 2);
        d1, d2, d3: control;
        a, b, c; array[1 .. 2, 1 .. 2] of control;
        procedure mu(x, y, z);
             entry —
                  begin
                       conn1.co(x, u[1,1]) → a[1,1];
                       conn2.co(x, u[1,2]) → a[1,2];
                       conn3.co(y, v[1,1]) → b[1,1];
                       conn4.co(y, v[2,1]) → b[2,1];
                       conn5.co(x, u[2,1]) → a[2,1];
                       conn6.co(x, u[2,2]) → a[2,2];
                       conn7.co(y, v[1,2]) → b[1,2];
                       conn8.co(y, v[2,2]) → b[2,2];
                  end;
                  for i := 1 to 2 do
                       for j := 1 to 2 do
                            a[i,j], b[i,j] →
                                 mult[i,j].mu(u[i,j], v[i,j], w[i,j])
                            → c[i,j]
                       od
                  od;
                  begin
                       c[1,1] →
                            for k := ∅ to N - 1 do
                                 r[k + N] := w[1,1,k] od;
                       c[2,2] →
                            for k := ∅ to N - 1 do r[k] := w[2,2,k] od;
                  end → d1;
```

```
                    begin
                         c[1,2]→ shift1.sh(w[1,2], s1);
                         c[2,1]→ shift2.sh(w[2,1], s2);
                    end→d2;
                    d2→ add1.ad(s1, s2, t) →d3;
                    d1, d3→add2.ad(r, t, z)→ return;
          end mu;
     assume N = 1;
          procedure mu(x, y, z);
               entry→
                    z[0] := x[0] and y[0]; z[1]:= false
               →return
          end mu;
     endassume;
end multiplier
```

5. Prinzipien für den TDSL-Compiler

Bisher standen die Gemeinsamkeiten zwischen Silicon-Sprache und Programmiersprache im Vordergrund der Betrachtung. Der Übersetzungsprozeß muß notwendigerweise erhebliche Unterschiede aufweisen. Ein Silicon-Compiler muß eine ungleich schwierigere Aufgabe bewältigen als ein Software-Compiler. Er muß kritische Platz-, Zeit- und Energiebedarfsrestriktionen erfüllen. Diese Restriktionen stehen jeweils in unterschiedlicher Weise, so daß für einen und denselben Typ an verschiedenen Stellen im allgemeinen ein unterschiedliches Ergebnis erzeugt werden muß. Die Verbindungsmoduln wie conn1 bis conn8 in unserem Beispiel sind alle vom selben Typ, aber je nach Anschlußbedingungen müssen sie unterschiedlich übersetzt werden.

Ferner ist die Verifikation des Ergebnisses ein wesentlich vielschichtigeres Problem als in der Programmierung, da die Parallelität der Abläufe und die Einhaltung von Zeit- und elektrischen Toleranzen berücksichtigt werden müssen. Diese Aufgabe wird bisher nur durch heuristisch gelenkten Einsatz verschiedener Analysetechniken bewerkstelligt.

Der für die Sprache TDSL adäquate Compiler muß eine top-down-Übersetzung realisieren. Das bedeutet: Beginnend mit dem Hauptmodul wird die Übersetzung eines Moduls jeweils auf die Übersetzung seiner Teilmoduln und die

Schaffung der Verbindungen zurückgeführt. Im Normalfall wird jeder Modul dort neu übersetzt, wo er vorkommt, auch wenn ein Modul gleichen Typs schon früher übersetzt wurde. So wird die Anpassung an die lokal bestehenden Einschränkungen berücksichtigt.
Mit dem top-down-Übersetzungsprozeß geht eine schrittweise Verfeinerung der Aufteilung der Chipfläche einher: Für jeden zu übersetzenden Modul wird ein Stück Chipfläche vorgegeben, an dessen Rand Anschlüsse plaziert sind. In dieser Fläche ist der Zielcode unterzubringen. Der voraussichtliche Platzbedarf eines Moduls wird bottom-up aus dem Platzbedarf seiner Teilmoduln berechnet. Wurde er unterschätzt, so ist der Modul nicht übersetzbar; es ist ein höherer Platzbedarf zu veranschlagen, was eine Neuübersetzung des übergeordneten Moduls erfordert.
Das zweite wichtige Merkmal ist die Lokalität der Datenübergabe. Durch Sichtbarkeitsbeschränkungen für die Objekte im TDSL-Text wird erzwungen, daß bei der Übersetzung eines Teilmoduls nicht unvorhergesehen eine Verbindung zu einem sehr weit entfernten anderen Teilmodul gezogen werden muß. Die Sichtbarkeitsbeschränkungen bewirken, daß stets nur Daten zwischen direkt über- oder nebengeordneten Objekten unmittelbar übergeben werden. Sind Schaltzeiten für die elementaren Vorgänge bekannt, so kann das Zeitverhalten schon aufgrund des Quelltextes grob abgeschätzt werden.

6. Probleme

Die Veröffentlichung eines Sprachkonzeptes kann, wie die Erfahrung lehrt, die Entwicklung von Compilern befördern. Im Fall der Silicon-Compiler ist der Weg allerdings besonders weit.
Einige Fragen, die außerhalb jeder Betrachtung geblieben sind, werden hier genannt:

1) das Problem der Aufteilung von Funktionen auf Hardwarekomponenten, z. B. die Zerlegung der Befehle eines programmierbaren Schaltkreises in Mikrobefehle, die jeweils von einer Komponente realisiert werden. Ganz allgemein gesprochen handelt es sich um das Problem der Mehrfachnutzung von Moduln unter dem Gesichtspunkt der Platzbeschränkung.
2) die Auswahl geeigneter Grundfunktionen, die (technologieabhängig ?) als leicht zu realisierende Funktionen zur bevorzugten Verwendung empfohlen werden,
3) das Problem der Unterscheidung zwischen Variablen, die einen sta-

tischen Speicher benötigen, und kurzzeitig belegten Variablen,

4) der Aufbau des Compilers aus Konstruktions- und Verifikationswerkzeugen

und vieles andere mehr.

7. Literatur

/1/ Daniel D. Gajski "THE STRUCTURE OF A SILICON COMPILER"
in Proc. IEEE Int. Conf. on Circuits aud Computers (New York) 1982

/2/ Stephen C. Johnson "VLSI CIRCUIT DESIGN REACHES THE LEVEL OF ARCHITECTURAL DESCRIPTION"
ELECTRONICS, May 3, 1984

/3/ John F. Buckley "HIGH-LEVEL HARDWARE DESCRPTION LANGUAGES: A NEW COMPUTER AIDED DESIGN TOOL"
British Telecommunications Engineering Journal, Vol 2, Part 4, Jan. 1984

/4/ Jeffrey D. Ullman "COMPUTATIONAL ASPECTS OF VLSI"
COMPUTER SCIENCE PRESS 1983

/5/ Peter Marwedel "THE MIMOLA DESIGN SYSTEM: A DESIGN SYSTEM WHICH SPANS SEVERAL LEVELS"
in: W. Giloi (ed): "METHODOLOGIES FOR COMPUTER SYSTEM DESIGN"
North Holland, 1984

/6/ Karl J. Lieberherr, Svend E. Knudsen "ZEUS: A HARDWARE DESCRIPTION LANGUAGE FOR VLSI"
Report, Eidgenössische Technische Hochschule Zürich, Institut für Informatik, Februar 1983

/7/ Jan Grabowski "GRUNDMERKMALE EINER QUELLSPRACHE FÜR TOP-DOWN-SILICON-COMPILER"
ORZ der Humboldt-Universität zu Berlin, 1984, unveröffentlicht

Dr. sc. Jan Grabowski
Dipl. Math. W. Dietmar Gellermann
Humboldt-Universität
1086 Berlin
Organisations- und Rechenzentrum
PSF 12 97

Grabowski, Jan
ORZ der Humboldt-Universität, DDR, 1086 Berlin, PSF 1297

Hypothesenbildung in wissensverarbeitenden Systemen: Thesen und Implementationsbeispiele in PROLOG

1. Hypothesenbildende Systeme

Wissensverarbeitende Systeme sollen über die Fähigkeit verfügen, aus gespeichertem Wissen (Fakten, Gesetzen) Schlüsse zu ziehen, also weiteres Wissen zu erzeugen. Häufig spielt dabei der Umstand eine Rolle, daß dem System unvollständige Informationen über einen Gegenstandsbereich mitgeteilt wurden, die diesen nicht systematisch, sondern nur exemplarisch beschreiben. Mitunter liegt das einfach daran, daß der Benutzer überhaupt nicht in der Lage ist, den Gegenstandsbereich systematisch zu beschreiben.

Um geforderte Funktionen, z. B. Abfrageaufgaben, auf der Basis unvollständigen Wissens erfüllen zu können, muß mitunter fehlendes Wissen durch eine Hypothese ersetzt werden. Ein wissensverarbeitendes System, das über eine solche Möglichkeit verfügt, nennen wir hypothesenbildend. Die Hypothesen können falsch und die Reaktionen des Systems daher fehlerhaft sein; es ist Sache des Benutzers, die Bildung falscher Hypothesen durch ein möglichst vollständiges Informationsangebot auszuschließen bzw. beim Erkennen von Fehlern das Angebot entsprechend zu ergänzen, d.h. das System zu "belehren". Beispielsweise können die exemplarischen Informationen Beispielsätze einer Sprache sein, deren Syntax dem System zwar in Grundzügen bekannt ist, aber in Einzelheiten durch Auswertung der Beispiele ergänzt werden soll.

Das System muß ggf. Hypothesen über syntaktische Regeln aufstellen, um einen Satz überhaupt formulieren zu können.

Ein anderes Beispiel: Die unvollständigen Informationen mögen aus Input-Output-Beispielen einer Funktion bestehen, die dem System nur bis auf Zugehörigkeit zu einer Klasse von Funktionen bekannt ist; das System muß ein hypothetisches Programm für die Funktion bilden, um weitere Funktionswerte vorherzusagen.

Man mache sich klar, daß die "Belehrung" im allgemeinen nicht darauf hinauslaufen darf, daß der Benutzer alle vom System gebildeten Hypothesen unmittelbar auf ihre Richtigkeit überprüft, denn diese können ihm unverständlich, zu kompliziert oder zu zahlreich sein. Wir betrachten daher den Fall der Belehrung durch Beispiele als den Normalfall.

Hypothesen können permanent, d.h. wie Fakten oder Gesetze zur weiteren Verwendung gespeichert, oder temporär sein. Eine temporäre Hypothese wird nur zur Erfüllung einer geforderten Abfrageleistung gebildet und anschließend wieder vergessen.
Permanente Hypothesen können mit neu hinzukommendem Wissen in Widerspruch geraten (inkonsistent werden); wenn wir solche Hypothesen als unzulässig betrachten, so bedeutet dies, daß nach jeder Erweiterung des gesicherten Wissens alle Hypothesen von neuem auf ihre Konsistenz überprüft werden müssen. Bei temporären Hypothesen entfällt dies.

Unten soll die Arbeit mit temporären Hypothesen an einem Beispiel in der Sprache PROLOG dargestellt werden.

2. Ein Beispiel

Wir stellen uns vor, daß das wissensverarbeitende System deutschsprachige Sätze analysieren, interpretieren und beantworten soll. Die Sätze mögen Operationen über einer sehr einfachen Datenbasis ausdrücken, die aus einer zweistelligen Relation "element" besteht. Die Aussage

"Die Katze ist ein Tier."

möge besagen, daß die Beziehung

element ("Katze", "Tier")

zutrifft. Ein korrekter Dialog mit dem System könnte so ablaufen:

Benutzer: "Die Katze ist ein Tier."
System: "Verstanden."
Benutzer: "Ist die Katze ein Tier?"
System: "Ja."
Benutzer: "Ist die Gemse ein Tier?"

System: "Nein."
Benutzer: "Gibt es ein Tier?"
System: "Ja."
Benutzer: "Wer ist ein Tier?"
System: "Die Katze ist ein Tier."

Richten wir unsere Aufmerksamkeit auf den Teil des Wissenserwerbs, der sozusagen neben dem inhaltsbezogenen Wissenserwerb nebenher läuft: den Erwerb der Fähigkeit, deutschsprachige Sätze zu analysieren und zu bilden. Wir gehen davon aus, daß die Konstruktionsregeln der vorkommenden Sätze bereits bekannt sind, die grammatikalischen Merkmale der vorkommenden Substantive aber durch Auswertung der Beispielsätze erst zu erlernen sind.
Spätestens nach der Auswertung des Satzes "Die Katze ist ein Tier" steht fest, daß "Katze" ein weibliches und "Tier" ein männliches oder sächliches Substantiv ist. Angenommen, das System läßt die unvollständige Information unberücksichtigt, speichert also "Tier" als Substantiv mit unbekanntem Geschlecht, so ist als Antwort auf die Frage "Wer ist ein Tier?" die syntaktisch falsche Antwort "Die Katze ist eine Tier" möglich. (Bildung der Hypothese, daß "Tier" weiblich sei.) Angenommen, das System speichert als permanente Hypothese, daß "Tier" männlichen Geschlechts sei, so wird es auf die Frage "Gibt es ein Tier?" vielleicht irrtümlich mit "Nein" antworten, da "ein Tier" in dieser Frage ersichtlich <u>nicht</u> die Form eines männlichen Substantivs hat, die erfragte Information jedoch im Speicher mit einem männlichen Substantiv verbunden ist. Es bleibt also nichts anderes übrig, als auch unvollständige Informationen wie die, daß "Tier" ein männliches <u>oder</u> sächliches Substantiv ist, zu verwerten.

3. Implementation des Beispiels

Wir geben nun eine PROLOG-Implementation für unsere Beispiel-Aufgabe an und achten dabei auf die Auswertung der unvollständigen Information und die Bildung temporärer Hypothesen.

Wir halten uns bezüglich PROLOG an die Terminologie aus /1/
Der Frage-Antwort-Zyklus kann wie folgt hingeschrieben werden:

```
(1)  zyklus: - prompot,          /* Eingabeanforderung */
             lies (Satz)
             interpretiere (Satz, Antwort),
             schreibe (Antwort).
```

(Die nicht in Anführungszeichen eingeklammerten Buchstabenketten sind PROLOG-Atome (-Konstanten), falls sie mit einem Kleinbuchstaben beginnen, und sonst Variable. Wir behalten die Kleinschreibung auch im Text bei und heben dort die Atome und Variablen durch Unterstreichung hervor.)
Die Prozedur <u>lies</u> liefere den gelesenen Satz in Form einer Liste aus Zeichenketten, also etwa ["die", "Katze", "ist", "ein", "tier"].
Die Prozedur <u>interpretiere</u> muß den Satz syntaktisch und semantisch analysieren, die darin geforderte Operation ausführen und einen Antwortsatz synthetisieren. Wir beginnen mit dem Interpretieren eines "Wer-ist-Satzes":

```
(2) interpretiere (Satz, Antwort)
      : - weristsatz (Satz, Klasse),
          element (Element, Klasse),
      bilde_istsatz (Antwort, Element, Klasse).
```

(Es wird zunächst festgestellt, ob <u>Satz</u> ein "Wer-Ist-Satz" ist, also ein Fragesatz, der nach Elementen einer Klasse <u>Klasse</u> fragt. Wenn ja, so wird mit Hilfe von <u>element</u> ein Element der Klasse gesucht und der Variablen <u>Element</u> zugewiesen. Schließlich wird ein Satz gebildet, der aussagt, daß <u>Element</u> ein Element von <u>Klasse</u> ist.)

```
(3) weristsatz (Satz, Klasse)
      : - verketten (["wer","ist"], Klassenteil, Satz)
          substantivphrase (Klassenteil, Klasse, nominativ,
          unbestimmt).
```

<u>Klassenteil</u> ist der auf ["wer", "ist"] folgende Teil von <u>Satz</u>. <u>Klassenteil</u> muß eine <u>substantivphrase</u> mit dem Substantiv <u>Klasse</u> und dem <u>unbestimmten</u> Artikel im <u>nominativ</u> sein:

(4) substantivphrase ([Wort 1, Wort 2] , Substantiv, Fall, Artikeltyp)
: - artikelform (Wort 1, Geschlecht, Fall, Artikeltyp),
substantivform (Wort 2, Substantiv, Fall),
not (falsches_Geschlecht (Substantiv, Geschlecht)).

(5) artikelform ("ein", maennlich, nominativ, unbestimmt).

(6) artikelform ("ein", saechlich, nominativ, unbestimmt).

(7) substantivform (Wort, Wort, nominativ).

Die Prozedur substantivphrase prüft nicht die exakte Übereinstimmung von Geschlecht mit dem Geschlecht des Substantivs, da dieses möglicherweise noch nicht eindeutig bekannt ist. Sie prüft lediglich, daß nicht "positiv bekannt" ist, daß Geschlecht n i c h t zum Geschlecht des Substantivs paßt. (Diese Information ist in falsches_Geschlecht gespeichert.)

(8) bilde_istsatz (Antwort, Element, Klasse)
: - verketten (Elementteil, ["ist" | Klassenteil] , Antwort),
substantivphrase (Elementteil, Element, nominativ, bestimmt),
substantivphrase (Klassenteil, Klasse, nominativ, unbestimmt).
artikelform ("die", weiblich, nominativ, bestimmt).

Bei bilde_istsatz wird vermöge der Prozedur substantivphrase darauf geachtet, daß die Geschlechter der verwendeten Artikel den Geschlechtern der verwendeten Substantive nicht bekanntermaßen widersprechen. Jede Artikelform, die diese Eigenschaft hat, ist als Hypothese zulässig, um den Satz zu bilden.
Wir behandeln nun die Auswertung eines "Ist-Satzes".

(9) interpretiere (Satz, Antwort)
: - analysiere_Istsatz (Satz, Element, Klasse),
merke (element (Element, Klasse)).

merke (Fakt) sei dabei eine Prozedur, die den Fakt Fakt speichert, falls er noch nicht gespeichert ist. Außer dem Fakt element (Element, Klasse) sollen jedoch auch die neuen Fakten über im Satz vorkommende Substantive gemerkt werden. Dies ge-

schieht in analysiere_Istsatz:

(10) analysiere_Istsatz (Satz, Element, Klasse)
: - verketten (Elementteil, ["ist" | Klassenteil] , Satz),
analysiere_Substantivphrase (Elementteil, Element, nominativ, bestimmt),
analysiere_Substantivphrase (Klassenteil, Klasse, nominativ, unbestimmt).

(11) analysiere_Substantivphrase ([Wort 1, Wort 2] , Substantiv, Fall, Artikeltyp)
: - substantivphrase ([Wort 1, Wort 2], Substantiv, Fall, Artikeltyp),
asp (Wort 1, Substantiv, Fall, Artikeltyp).

(12) asp (Wort, Substantiv, Fall, Artikeltyp)
: - not (artikelform (Wort, Geschlecht, Fall, Artikeltyp)),
merke (falsches_Geschlecht (Substantiv, Geschlecht)),
fail.

(13) asp (Wort, Substantiv, Fall, Artikeltyp).
fail bedeutet: Gehe zurück und probiere eine andere Regel. In unserem Falle heißt das: Suche einen weiteren Wert von Geschlecht, für den die Aussage
not (artikelform (Wort, Geschlecht, Fall, Artikeltyp))
erfüllt ist, und merke auch hierfür den Fakt falsches_Geschlecht (Substantiv, Geschlecht).
Sind alle Möglichkeiten erschöpft, so tritt die zweite Regel für asp in Aktion, die zur sofortigen Beendigung der Prozedur asp führt.

4. Gründe für eine Erweiterung der Sprache PROLOG

Wie das obige Beispiel zeigt, eignet sich PROLOG sehr gut zur Programmierung des Erzeugens temporärer Hypothesen; als Hypothesen treten die während des Abarbeitens einer Anweisung temporär erzeugten Variablenbelegungen auf. Die Hypothese muß dabei Kontrollprozeduren passieren, die die Konsistenz prüfen; besteht eine Hypothese die Prüfung nicht, wird sie verworfen und sogleich die nächst probiert.

Als Kontrollprozedur tritt z. B.

not (falsches_Geschlecht (Substantiv, Geschlecht))

auf.

Die Relation falsches_Geschlecht wird aus Beispielen aufgebaut, z. B. liefert die Analyse von "Die Katze ist ein Tier" die Fakten

(14) falsches_Geschlecht ("katze", maennlich).

(15) falsches_Geschlecht ("katze", saechlich).

(16) falsches_Geschlecht ("tier", weiblich).

Die Erfahrung lehrt, daß die Auswertung unvollständiger Informationen im Regelfall zu ähnlichen Konstruktionen führt. Dem Wesen nach handelt es sich nämlich um Ungleichungen, wie wir sehen werden.

Wir probieren eine andere Schreibweise für diese Konstruktion aus: Wir notieren die Regel (4) in der Form

(4') substantivphrase ([Wort 1, Wort 2], Substantiv, Fall, Artikeltyp)

: - artikelform (Wort 1, ∂geschlecht (Substantiv), Fall, Artikeltyp),

substantivform (Wort 2, Substantiv, Fall).

und die Fakten (14) - (16) in der Form

(14') ∂geschlecht ("katze") ≠ weiblich.

(16') ∂geschlecht ("tier") ≠ weiblich.

Hier bezeichnet der mit ∂ beginnende Name ein in PROLOG gewöhnlich nicht vorkommendes Objekt, nämlich eine Funktion. Funktionen sind zu interpretieren als Abbildungen über der Menge der PROLOG-Terme. Die Gleichung (11') und die Ungleichung (13') besagen, daß die Funktion ∂geschlecht, angewendet auf den PROLOG-Term "katze", den PROLOG-Term weiblich liefert, dagegen bei Anwendung auf den PROLOG-Term "tier" einen von weiblich verschiedenen PROLOG-Term.

Ein Term, dessen Funktor eine Funktion bezeichnet, möge mit einem zweiten Term genau dann unifizierbar sein, wenn die Ungleichheit beider nicht beweisbar ist.
Demnach ist

∂geschlecht("tier")

mit

maennlich

unifizierbar, weil die Ungleichung

∂geschlecht("tier") ≠ maennlich

nicht aus den bekannten Ungleichungen und Gleichungen herleitbar ist. Also

substantivphrase (["der", "tier"], "tier", nominativ, bestimmt)

ist erfüllbar; dagegen ist

substantivphrase (["ein", "katze"], "katze", nominativ, unbestimmt)

nicht erfüllbar.
Diese Erweiterung der Sprache PROLOG, die wir hier nicht ganz präzise und vollständig beschrieben haben (insbesondere sind wir den Ableitungsbegriff für Termungleichungen schuldig geblieben), paßt die Sprache noch besser den Erfordernissen der Auswertung unvollständiger Informationen an. Dies ergibt sich aus der großen Bedeutung, die Termungleichungen für die Hypothesenbildung beim Schließen aus unvollständiger Information haben (vgl. Jantke /2/).

5. Praktische Erfahrungen

Am ORZ der Humboldt-Universität werden z. Zt. mit dem PROLOG-Interpreter IIUW-Prolog gesammelt. (IIUW-Prolog wurde am Institut für Informatik der Universität Warschau in PASCAL 360 für ESER-Anlagen programmiert.) Die vorliegenden Erfahrungen betreffen die Arbeit mit Datenbanken, die deutschsprachige Konversation und insbesondere das Schließen aus unvollständigen Informationen. Es ist beabsichtigt, die in 4. vorgeschlagene Erweiterung der Sprache an diesem Interpreter vorzunehmen und zu erproben.

/1/ Clocksin, W. F./Mellish, C. S.: Programming in Prolog.
Springer-Verlag New-York 1981

/2/ Jantke, K. P. The main proof theoretic problems in inductive inference.
In: 2nd Frege Conference 1984
G. Wechsung (Ed.)
Akademie-Verlag Berlin 1984

Hartwig,M.; Strobel,R.; Stein,E.

Über die Implementierung einer Teilmenge von Ada

Die Programmiersprache Ada hat als mögliche Basis einer einheitlichen Software-Technologie starke Beachtung gefunden.

Am ZKI der ADW wird gegenwärtig an der Implementierung einer Teilmenge von Ada (Ada-0) gearbeitet.Der entstehende Ada-0 Compiler ist als Ausgangspunkt für einen in Ada programmierten Übersetzer der vollständigen Sprache geplant. Dieser Ansatz soll es ermöglichen, zusammen mit der Entwicklung und Testung der Übersetzungsalgorithmen auch Programmiererfahrungen in Ada zu sammeln.

Im Vortrag wird versucht, anhand von Problemen des Entwurfs und der Programmierung des Ada-0 Compilers Einblick in den 'Ada-Stil' zu geben.

Hopfer, Reiner

High-Level Language (HLL-)-Rechnerverbund

1. Allgemeines

Die Entwicklung der Programmierung in Richtung der überwiegenden Anwendung von höheren Programmiersprachen erfordert eine zweckentsprechende Strukturierung der Hardware sowie die Bereitstellung geeigneter Systemsoftware für Rechner und Rechnerverbundsysteme. Das dabei verfolgte Ziel ist zum einen eine höhere Leistungsfähigkeit der rechentechnischen Geräte und zum anderen die Schaffung von Möglichkeiten für eine effektivere und zuverlässigere Softwareentwicklung. Darüber hinaus macht sich infolge der immer komplexer werdenden Aufgabenstellungen für den Nutzer eine Vereinfachung der Programmierung, insbesondere von parallelen Prozessen und Echtzeitvorgängen, notwendig. Gegenwärtig zeichnen sich daher im internationalem Maßstab Entwicklungen ab, die eine sogenannte direkte Ausführung von Programmen in Hochsprache zum Inhalt haben. Das bisher angewendete Übersetzer-Binde-Lade-Prinzip ist nicht mehr notwendig. Rechner, die eine derartige Arbeitsweise gestatten, werden als HLL-Rechner (High-Level Language-Rechner) bezeichnet /1,2,3/. Der Vorteil eines solchen vereinfachten Gebrauchs von Rechnern wird besonders wirksam, wenn HLL-Rechner zu HLL-Rechnerverbundsystemen verknüpft werden.

2. High-Level-Sprachen

High-Level (HL-)-Sprachen ("high-level" im Sinne hoch oder "weit weg" von der Binärdarstellung herkömmlicher Maschinensprachen) sind höhere Programmiersprachen, die die Darstellung von Informationsverarbeitungsprozessen auf einem relativ hohen Abstraktionsniveau gestatten und insbesondere solche Eigenschaften aufweisen, die die maschinelle Interpretation eines entsprechenden Programmes ermöglichen. Eine allen Ansprüchen gerecht werdende HL-Sprache gibt es z. Z. noch nicht.
HL-Endsprachen sind für den Nutzer gedacht, während HL-Zwischensprachen die Funktion einer internen Bezugssprache übernehmen (Bild 1). Unter den HL-Endsprachen haben neben den "normalen"

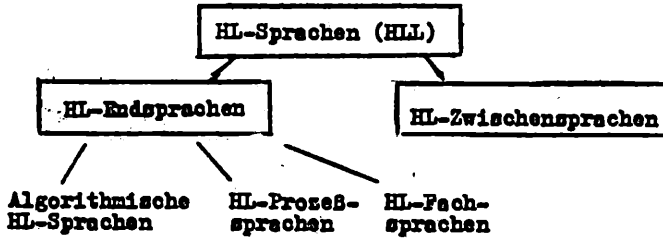


Bild 1: Klassifizierung der HL-Sprachen

(algorithmischen) HL-Sprachen insbesondere HL-Prozeßsprachen eine größere Bedeutung. Hier sind sämtliche für einen Echtzeitbetrieb notwendigen Sprachelemente wie für den Parallellauf von Echtzeitaufträgen, Ereignissteuerung, Prioritätsscheduling und zeitliche Synchronisation zwischen den Systemaktionen und dem technischen Prozeß in der Sprache enthalten.

3. HLL-Rechner

HLL-Rechner sind Hard- und/oder Softwaresysteme, die in HL-Sprachen formulierte Anwenderprogramme direkt ausführen können. Im allgemeinen ist bei einer solchen Betriebsweise kein für den Nutzer zugängliches Betriebssystem im herkömmlichen Sinne vorhanden. Außerdem sind keine Übersetzungen durchzuführen. Alle Schritte der Programmierung und die für die Steuerung des Informationsverarbeitungsprozesses notwendigen Aktionen werden mit den Mitteln der HL-Sprache beschrieben. Eventuell fehlerhafte Sprachkonstruktionen oder Laufzeitfehler werden während der Ausführung eines Programmes erkannt und angezeigt. Im Bild 2 sind die möglichen Prinzipien der Sprachverarbeitung dargestellt, wobei im klassischen Fall die Zwischensprache auch entfallen kann. Die direkte Interpretation einer HL-Endsprache entspricht der tatsächlichen HLL-Verarbeitung. Demgegenüber muß bei der Interpretation einer HL-Zwischensprache ein Übersetzungsvorgang vorgelagert werden. Der gesamte Prozeß wird auch Quasi-HLL-Verarbeitung bezeichnet. Die Vorteile einer HLL-Verarbeitung bestehen in einem effektiveren Abarbeiten der Programme, dem

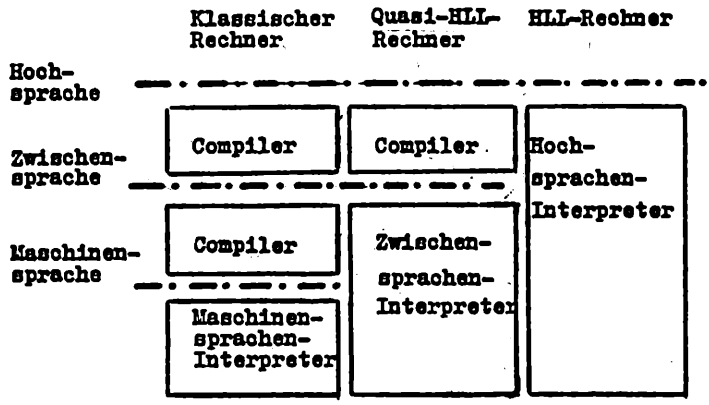


Bild 2: Verarbeitungsprinzipien bei Digitalrechnern

Wegfall der sonst üblichen maschinellen Zwischenstationen eines Programmes, der leichteren Fehlerermittlung und -korrektur sowie einer höheren Softwarezuverlässigkeit. Als Nachteil ist zu werten, daß HLL-Systeme im allgemeinen nur für eine Programmiersprache optimal arbeiten und der Einsatz bereits vorhandener Software aus anderen Quellen begrenzt ist.

4. HLL-Rechnerverbund

4.1. Physische Grundstrukturen

Komplexe HLL-Systeme werden durch Verbund von HLL-Prozessoren bzw. HLL-Rechnern (Verarbeitungseinheiten) auf dem Niveau einer HL-Sprache aufgebaut. Sämtliche auszutauschenden Informationen werden mit den Mitteln der HL-Sprache dargestellt. Beim Verbund von HLL-Verarbeitungseinheiten haben drei Grundstrukturen besondere Bedeutung (Bild 3):

- Multi-HLL-Prozessorsysteme
- Multi-HLL-Rechnersysteme
- HLL-Rechner-Netzverbund.

Während die ersten zwei Systemklassen insbesondere für homogene Systemkomponenten und nur für kleinere Entfernungen (lokaler

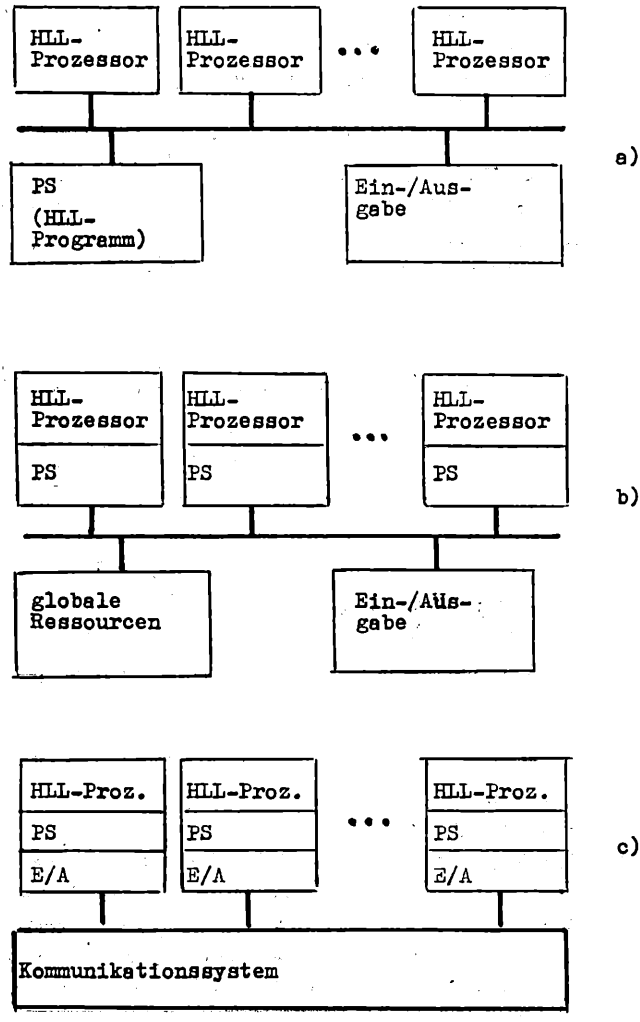


Bild 3: Physische Strukturen bei HLL-Rechnerverbundsystemen (a) Multi-HLL-Prozessorsystem, b) Multi-HLL-Rechnersystem, c) HLL-Rechner-Netzverbund; PS - Programmspeicher, E/A - Ein- und Ausgabesystem)

Bereich) Anwendung finden, ist letztere mehr für die Verkopplung von heterogenen HLL-Komponenten sowie für kleine und große Entfernungen geeignet. Beim HLL-Rechner-Netzverbund ist das Kommunikationssystem die physische Basis aller notwendigen Informationsübertragungen zwischen den einzelnen HLL-Komponenten. Eine geeignete Lösung stellt das für herkömmliche Rechnernetze vielfach verwendete ISO-Referezmodell mit Schichtenstruktur dar, wobei der HLL-Verbund der obersten Anwendungsschicht entspricht /4/. Die physischen Strukturen von HLL-Rechnerverbundsystemen sind jedoch für den Nutzer nicht primär von Interesse, sondern mehr die darauf abzubildenden logischen Strukturen.

4.2. Logische Strukturen

Bei HLL-Verbundsystemen kann der Nutzer entsprechend seiner zu lösenden Aufgabenstellungen logische Strukturen von Verarbeitungseinheiten und von zu bearbeitenden Dateneinheiten bzw. -beständen aufbauen. Besondere praktische Bedeutung in der Organisation der Verarbeitungsprozesse haben Gleichordnungsstrukturen, Unterordnungsstrukturen und hierarchische Strukturen (Bild 4). Hinsichtlich der (logischen) Kopplungsart hat man

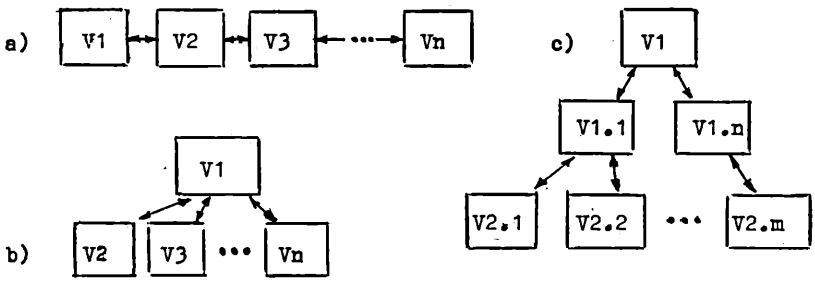


Bild 4: Ausgewählte logische Strukturen bei HLL-Rechnerverbund
(a) Gleichordnungsstruktur, b) Unterordnungsstruktur,
c) Hierarchische Struktur; V - HLL-Verarbeitungseinheit)

zu unterscheiden:

- Direktkopplung
  Die Anweisungen werden ausgetauscht und anschließend von der empfangenden Verarbeitungseinheit sofort ausgeführt.
- Indirektkopplung
  Anweisungen und Daten werden übertragen, zu einem Programmmodul zusammengestellt und erst dann ausgeführt.

4.3. Dialogbetrieb, Prozeßbetrieb

Da im allgemeinen ein HLL-Rechnerverbundsystem für den Nutzer kein "sichtbares" Betriebssystem enthält, müssen bei Dialogbetrieb bzw. interaktiven Betriebsweisen die entsprechenden Dialogkomponenten im Sprachverarbeitungssystem enthalten sein. In analoger Weise gilt bei Prozeßbetrieb, daß im Sprachsystem Echtzeitkomponenten enthalten sind, die die Synchronisation und die Kommunikation von Programmeinheiten sowie die Ereignissteuerung und die Zeitverwaltung übernehmen. Alle zu realisierenden Aktionen sind demnach mit den Mitteln der HL-Sprache auf logischem Niveau zu beschreiben. Dies betrifft insbesondere auch logische Interrupts und logische Unterbrechungsbehandlungsroutinen /5/.

5. Praktische Realisierung

Für die Realisierung eines HLL-Rechnerverbundes wird ein hierarchisches Rechnersystem mit drei Ebenen vorausgesetzt, was in physischer und logischer Struktur übereinstimmt /6/. Die logische Verkopplung erfolgt auf dem Niveau einer HL-Endsprache. Als Ausgangssprache dient die Programmiersprache BASIC, die durch entsprechende Sprachelemente erweitert wird. Eine Echtzeitfähigkeit wird durch zusätzliche Sprachelemente auf der untersten Hierarchieebene erreicht.

6. Zusammenfassung

Im Vortrag werden der Rechnerverbund von mehreren HLL-Rechnern auf dem Niveau einer höheren Programmiersprache behandelt, ausgewählte Lösungsmöglichkeiten angegeben sowie deren Vor- und Nachteile diskutiert. Es wird gezeigt, welche sprachlichen Mittel zur Realisierung eines Informationsaustausches zwischen den verschiedenen Rechnern notwendig und wie die einzelnen Sprachver-

arbeitungssysteme zu koppeln sind. Ein HLL-Rechnerverbund ist dann vorteilhaft einzusetzen, wenn dem Nutzer ein "durchsichtiges" verteiltes Rechnersystem zur Verfügung steht, auf dem er selbständig, ohne Inanspruchnahme zusätzlicher Betriebssystemdienste, auf einer relativ "hohen Ebene" seine Aufgabenstellung programmieren und die globale Gesamtfunktion des verteilten Systems überblicken kann. Besonders günstig erweist sich gegenwärtig ein Einsatz bei Verbund von mehreren Einchiprechnern sowie bei einem hierarchischen Verbund von Rechnern sehr unterschiedlicher Leistungsfähigkeit.

7. Literatur

/1/ W. K. Giloi: Rechnerarchitektur - Berlin, Heidelberg, New York: Springer-Verlag, 1981

/2/ Grimsdale, R. L.: Computer architektur for the interpretation of High-Level Languages; Informatik-Fachberichte, Computer Architektur: Berlin, Heidelberg, New York: Springer-Verlag, 1975, Bd. 4, S. 23-45

/3/ Mackrodt, W.: Gerätetechnisch unterstützte Zwischensprachen-interpretation, Hefte des Weiterbildungszentrum für MKRI, TU Dresden, 1981 Probleme der Rechnerarchitektur, Nr. 51 S. 29-43

/4/ Proft Proposal ISO IDP, "Data Processing - Open System interconnection - Basic Reference Model", ISO 97/16 N227, Dec. 1980

/5/ Hopfer, R.: Echtzeitprogrammierung mit höheren Programmiersprachen bei Mikrorechnersystemen, Hefte des Weiterbildungszentrums für MKRI, TU Dresden, 1985, Nr. 78 (im Druck).

/6/ Hopfer. R.: Ludwigs, K.; Reimann, D.: Hierarchischer Rechnerverbund, Fo.-Bericht, TH Leipzig, SMR, 1985

Autor:
Dr.-Ing. R. Hopfer
Technische Hochschule Leipzig
Sektion Mathematik und Rechentechnik
Informationsverarbeitung 2
7030 Leipzig
Karl-Liebknecht-Str. 132

Hutschenreiter, Jörg: Grundlagen und Implementation von verbundenen Fachsprachen

Mit zunehmendem Einsatz der Rechentechnik in verschiedenen Fachgebieten erhalten Fachsprachen als effektives Verständigungsmittel mit dem Automaten immer größere Bedeutung. Während mit bisherigen Compiler-Generatoren Compiler für algorithmische Programmiersprachen vom Spezialisten in relativ kurzer Zeit hergestellt werden können, verlangt das massenhafte Auftreten von Fachsprachen, daß diese bereits vom versierten Programmierer implementiert werden können.
Mit dem hier vorgestellten Forschungsresultat "Verbundene Fachsprachen" und seiner praktischen Umsetzung in Form des Metasystems DEPOT2a steht ein solches Werkzeug zur Nutzung zur Verfügung.
Es ist ausgerichtet auf die umfassende Verwirklichung fachsprachlicher Kommunikation (deskriptive und algorithmische Fachsprachen) Mit seiner Hilfe wird Software für die Analyse und Verarbeitung (z.B. Transformation in Zielsprachen, Interpretation) von Fachsprachen automatisch aus der Sprachbeschreibung mit Hilfe einer Metasprache erzeugt. Das Konzept der lokalen Sprachen gestattet es, Querverbindungen zwischen Fachsprachen herzustellen, bereits implementierte Fachsprachen und deren Teile nachzunutzen für andere Implementationen sowie eine schrittweise modulare Software Entwicklung vorzunehmen. Damit ist das System als Bestandteil einer effektiven Software-Technologie, die auf Fachsprachen aufbaut, einsetzbar.
Das System DEPOT2a ist sowohl für ESER- als auch SKR-Rechentechnik realisiert mit einheitlicher Metasprache ML2a und Steuersprache CL2a für das DEPOT-Betriebssystem.

Dr. R. Kosciolowicz

Erfahrungen bei der Anwendung des Modulkonzeptes von M-PASCAL

1. Modulkonzepte in verschiedenen Programmiersprachen

Der Begriff des Moduls wird in sehr unterschiedlicher Weise verwendet und umfaßt je nach Anwendungsfall ein zusammenhängendes Programmstück, eine Prozedur, eine Übersetzungseinheit, eine Ladeeinheit oder vergleichbare Einheiten. Dadurch kommt es häufig zu Mißverständnissen bei der Benutzung dieses Begriffes. In diesem Beitrag geht es nicht um eine weitere Definition des Begriffes Modul bzw. modulare Programmierung sondern um Moduln im Sinne von MODULA 2 /4/ bzw. ADA und um das Programmieren mit solchen Moduln. Dabei handelt es sich nicht um ein Wortspiel. Moduln in diesem Sinne ermöglichen eine exakte Schnittstellenbeschreibung für die Zusammenarbeit miteinander, die auch vom Compiler überprüft werden kann. Es ist möglich, abstrakte Datentypen mit Spezifikationsteil (was wird gemacht?) und Implementationsteil (wie wird es realisiert?) zu entwerfen und exakt abzubilden. Für die gesamte Strukturierung und den Ablauf von Problemlösungen ergeben sich neue Möglichkeiten mit exakteren Kontrollmechanismen. Zu den bekannten Strukturierungsmöglichkeiten in Programmiersprachen mittels Prozeduren, strukturierten und elementaren Anweisungen kommen weitere wesentliche Formen hinzu. Mit Hilfe der Sprachkonstruktionen für Moduln ist es möglich, abstrakte oder reale Datentypen geeignet zu strukturieren und die Einheit von Datenobjekten und den zugehörigen Operationen sinnvoll auszudrücken. Durch getrennte Formulierung und Übersetzung von Definitions- und Implementationsteil auch zu verschiedenen Zeitpunkten können Entwurfsentscheidungen besser dokumentiert bzw. detaillierte Entscheidungen hinausgeschoben werden. Durch die separate Übersetzbarkeit von Moduln verbunden mit einer Kontrolle der exakten Schnittstelleneinhaltung stehen wesentlich bessere Hilfsmittel zur Verfügung als externe Prozeduren und gemeinsam genutzte globale Bereiche. Eine

deutliche Analogie der Modulprinzipien zu Entwurfsmethoden und -hilfsmitteln, die sich bereits bei der Konstruktion von Maschinen und Geräten durchgesetzt haben, ist nicht nur Zufall sondern Absicht. Mit dem immer weiteren Anstieg der Menge und der Komplexität der zu entwickelnden Software ist es notwendig, solche neuen und leistungsfähigen Sprachelemente für die allgemeine Nutzung bereitzustellen und Erfahrungen bei deren Anwendung zu sammeln.

2. Erfahrungen bei der Programmierung mit Moduln

Die bereits genannten Vorteile der Programmierung mit Moduln zeigen aber auch gleichzeitig das Hauptproblem - wie findet man eine gute Modulstruktur, die der Aufgabenstellung entspricht, einen hohen Verständlichkeitsgrad hat und spätere Modifikationen erleichtert? Es ist immer möglich, auch mit gut durchdachten, leistungsfähigen Sprachelementen einer Programmiersprache beliebig schlecht strukturierte und schwer verständliche Programme zu schreiben. Andererseits sind gut strukturierte Programme auch mit den Mitteln der Assemblersprache möglich. Dazu sind ein guter Programmierstil und umfangreiche Erfahrungen des Programmierers notwendig. Die Sprachelemente alleine sind nur ein Hilfsmittel, bestehende problemspezifische Zusammenhänge besser auszudrücken. Der Programmierer muß es lernen, mit diesen Hilfsmitteln effektiv umzugehen. Dies gilt besonders für die Strukturierungsmöglichkeiten, die deutlich über die Prozedurtechnik hinausgeht. Das Prinzip der Lokalität bei Daten und den zu ihnen gehörenden Operationen ist wesentlich besser realisierbar. Es bieten sich ausreichende Möglichkeiten zum Schutz von Implementationsbesonderheiten bei Datenstrukturen und Algorithmen (Information hiding). Durch eine bausteinartige Zerlegung in relativ selbständige Teilaufgaben kann die Komplexität der Problemlösung vereinfacht werden, wobei das exakte Zusammenspiel durch den Compiler und in der Laufzeit geprüft wird. Es ist möglich, universell nutzbare, problemspezifische Datentypen zu entwickeln und zur allgemeinen Nutzung bereitzustellen. Ein wesentliches Prinzip bleibt auch weiterhin gültig. Eine gute Struktur der entstehenden Software kann nicht als Ergeb-

nis vieler Modifikationen gefunden werden, sondern ist möglichst früh zu entwerfen. Dabei ist eine vollständige Umstrukturierung einer schlechten Struktur im Endeffekt besser und sicherer als mehrere Veränderungen an Teilstrukturen. Die Güte der gewählten Strukturierung erweist sich in der Regel erst beim Test und bei nachträglich notwendigen Modifikationen. Die Erfahrungen der letzten Jahre zeigen aber, daß auch die Einführung dieser neuen Sprachelemente in einen großen Nutzerkreis auf einen deutlichen traditionellen Widerstand stoßen wird. Deshalb ist es notwendig, anhand von Beispielen und ausgewählten Projekten die Leistungsfähigkeit dieser Ausdrucksmittel zu dokumentieren. Ihre effektive Anwendung erfordert aber ebenso Kreativität und Erfahrungen der Programmierer beim Umgang damit. Es muß sich ein klareres Denkschema bei der Strukturierung und Manipulation von Daten entwickeln. Außerdem zeigt sich, daß die Strukturierung des Programmablaufs und der Daten im gegenseitigen Wechselspiel vorzunehmen sind, um zu gut strukturierten und zuverlässigen Programmen zu kommen. Das Infragestellen der bisherigen Arbeitsweise bedeutet zwar eine Verunsicherung, ist aber auch gleichzeitig eine Quelle für neue Möglichkeiten und erfordert Geduld und Fingerspitzengefühl. Administrative Maßnahmen zur schnellen Einführung neuer Sprachkonstruktionen führen selten zum Erfolg, sondern verhärten oft bereits vorhandene Vorurteile.

3. Kriterien für die Strukturierung mit Moduln

Folgende Kriterien haben sich für die Wahl einer guten und verständlichen Struktur in der Literatur /4/ und bei der eigenen Arbeit bewährt:

- das Interface zwischen Moduln sollte möglichst einfach sein (Anzahl der gemeinsamen Größen und Komplexität der Zusammenhänge)
- kurze Importlisten sind anzustreben, besonders in Definitionsmoduln
- es sollten möglichst keine Variablen exportiert werden. Wenn Variable importiert werden, dann sollten sie nur als read-only-Variable benutzt werden (entspricht dem Seiten-

effekt bei Funktionen und Prozeduren)
- lokale Moduln (in Moduln oder in Prozeduren) sollten nur in Ausnahmefällen (problemspezifisch) angewendet werden
- Moduln sollten weitgehend unabhängig von der Programmumgebung entwickelt und getestet werden, in die sie eingebettet sind.

Es sind häufig 3 typische Modularten anzutreffen:

a) Aktionsmoduln, die im wesentlichen aus Prozeduren bestehen und meist nur Hilfsgrößen als lokale Daten haben. Sie werden z.B. für Konvertierungen zwischen verschiedenen Darstellungsformen (Liste, Baum, sequentielles File) oder für spezielle Algorithmen (Sortieren) benutzt.

b) Datenobjektmoduln, die einem konkreten Vertreter eines abstrakten Datentyps entsprechen. Es werden nur die Zugriffsoperationen nicht aber der interne Datentyp exportiert (z.B. Identifierliste, Fileverzeichnis).

c) Datentypmoduln, von denen mehrere Vertreter erzeugbar sind, da sowohl der Datentyp als auch die Zugriffsoperationen exportiert werden (z.B. Warteschlange, Keller).

Bei konsequenter Anwendung der angegebenen Möglichkeiten läßt sich z.B. ein Mißbrauch von Kenntnissen über den internen Aufbau von Datenstrukturen (z.B. Listen) verhindern, da der Zugriff nur über die exportierten Prozeduren und Funktionen erfolgen kann. Änderungen von Implementationsdetails, wie z.B. die Ersetzung eines Array durch eine verkettete Liste sind möglich, ohne die Benutzungsstellen außerhalb des Moduls zu verändern. Außerdem vereinfacht sich die Lokalisierung von Fehlerquellen sowie deren Beseitigung. Durch einen vorgezogenen Test herausgelöster Moduln läßt sich die Komplexität vieler Testfälle vereinfachen. Die nach den oben genannten Prinzipien entwickelten Programmen zeigen deutliche strukturelle Unterschiede zu herkömmlichen Programmen.

4. Erfahrungen bei der Arbeit mit M-PASCAL

Bei der Anwendung des Modulkonzeptes in M-PASCAL zeigten sich folgende Besonderheiten:
- Es existeren deutlich weniger Verschachtelungen von Moduln oder Prozeduren ineinander. Dafür gibt es mehr gleichbe-

rechtigte Moduln, deren Zusammenwirken durch Import-Export-Listen beschrieben wird (ähnlich dem Bausteinprinzip in technischen Anlagen).

- Es gibt keine langen und unübersichtlichen Vereinbarungsteile mehr, da entsprechend dem Lokalitätsprinzip Daten und zugehörige Aktionen in Moduln zusammengefaßt werden.
- Durch die Benutzung von Definitionsmoduln entstehen gleichzeitig wichtige Dokumentationselemente für die Struktur und die Schnittstellen untereinander.
- Definitionsmoduln sind eine wichtige Grundlage für Abstimmungen im Entwicklerkollektiv und trennen wesentliche von unwesentlichen Informationen.

Die von und verwendeten Compiler und Precompiler für M-PASCAL verfügen zur Zeit nicht über Möglichkeiten zur separaten Übersetzung von Definitions- und Implementationsmoduln. Damit ergaben sich einige technologische Probleme für die Übersetzung und den Test, die aber die Möglichkeiten der Anwendung des Modulkonzeptes für unser Projekt nicht wesentlich einschränkten. Umfangreiche Anwendungen von MODULA 2 für verschiedene Projekte in unserem Bereich lieferten vergleichbare Aussagen.

Literatur:

/1/ Bothe, K., Kosciolowicz, R. "M-PASCAL - a language for modular PASCAL programming", Preprint, Humboldt-Universität zu Berlin, 1983

/2/ Hohberg, B. "Abstrakte Datentypen beim Programmentwurf am Beispiel eines syntaxorientierten Programmeditors für PASCAL", WBZ TU Dresden, Heft 61, 1982

/3/ Kosciolowicz, R. "Konzeption eines Programmierlabors zur Unterstützung des modularen Programmentwurfs mit PASCAL", BIT'82 Tagungsmaterialien

/4/ Wirth, N. "Programming in MODULA 2", Springer Verlag Berlin Heidelberg New York, 1982

Dr. rer. nat. R. Kosciolowicz
Humboldt-Universität zu Berlin
Sektion Mathematik
1086 Berlin, Postfach 1297

Nager, Klaus

Anwendungen des Fachsprachensystems DEPOT

1. Das Fachsprachensystem DEPOT

Die Bedeutung von Fachsprachen, d.h. stark problemsbhängigen Programmiersprachen, als wesentlicher Bestandteil einer effektiven Software-Technologie ist in den letzten Jahren allgemein anerkannt worden. Diesem Trend tragen die Entwicklungen an den Varianten des Fachsprachensystems DEPOT (siehe /1/,/3/,/4/,/7/) Rechnung.
Die Hauptaufgabe dieses Programmsystems, das mit einer Metasprache zur Beschreibung der Syntax und der Verarbeitung von Sprachen ausgestattet ist, besteht in der automatischen Generierung von Fachsprachenübersetzern aus der metasprachlichen Beschreibung. Das System DEPOT2a eignet sich auch zur interpretativen Verarbeitung von Fachsprachen sowie zur Implementierung von Fachsprachenverbunden. Weiterhin wurde es bereits erfolgreich für Übertragungen von Programmen einer höheren Programmiersprache von einer Rechenanlage auf eine andere (Übersetzung zwischen verschiedenen höheren Programmiersprachen oder verschiedenen implementierungsabhängigen Varianten ein und derselben Sprache) sowie zur Formelmanipulation angewandt.
Um einen Eindruck von der Mannigfaltigkeit der mit DEPOT realisierbaren Fachsprachen zu vermitteln, sollen im folgenden drei der in jüngster Zeit entwickelten und implementierten Fachsprachen vorgestellt werden, und zwar M-FORTRAN, GEKO und DAPRUE. Daneben gab und gibt es eine Reihe weiterer bedeutender Anwendungen wie die Realisierung der Sprache PILOT zur Manipulation von Datenbanken oder Sprachen zur Robotersteuerung.

2. M-FORTRAN zur sprachlichen Stützung des Matrixmoduls ES1055/MAMO

Der Matrixmodul ES1055/MAMO wurde als Zusatzeinheit der Rechenanlage EC1055 zur beschleunigten Abarbeitung von Vektoroperationen geschaffen. Damit steigt durch Ausnutzung von Parallelarbeit die Abarbeitungsgeschwindigkeit von Vektoroperationen wesentlich gegenüber der komponentenweisen (skalaren) Verarbei-

tung durch die Zentraleinheit. Da die Programmierung dieses Prozessors auf sehr niedrigem Maschinensprachenniveau erfolgt, ist es als Voraussetzung für eine breite Nutzung des Matrixmoduls unbedingt erforderlich, eine sprachliche Stützung auf höhersprachlichem Niveau einzuführen.
Die an der Sektion Mathematik der TU Dresden entwickelte Sprache M-FORTRAN (siehe /5/) stellt eine echte Erweiterung von FORTRAN /ESER dar, in der Algorithmen über ein- und zweidimensionalen Feldern problemangepaßt, d.h. unter anderem mittels Operationen mit ganzen Feldern entsprechend dem in der Mathematik üblichem Matrizenkalkül, formuliert werden können. Der mit DEPOT2a realisierte Übersetzer ist in der Lage, Teiloperationen über diesen Feldern dem schnelleren Matrixmodul zu übertragen.

Im einzelnen sind in M-FORTRAN folgende Feldoperationen vorhanden:

s.*.A	Multiplikation des Feldes A mit skalarem Faktor s
A^{+}_B, A.*.B	elementweise Addition, Subtraktion, Multiplikation der Felder A und B
A*B	Skalarprodukt bzw. Matrizenprodukt
A(a:b)	Teilfeldbildung, a unterer, b oberer Index
A(*,k), A(i,*)	Teilfeldbildung, k-te Spalte bzw. i-te Zeile der Matrix A
A**k	Potenzieren einer Matrix A
A*T	Transponieren
A**-1	Invertieren

Beliebige Klammerungen in den Feldausdrücken sind möglich. Felder können auf Gleichheit geprüft werden. E bezeichnet die Einheitsmatrix, 0 die Nullmatrix. Überlagerte Teilfelder können mit neuen Variablennamen bezeichnet werden. Weiterhin unterstützt die Kennzeichnung einer Matrix bei der Deklaration als symmetrische, antisymmetrische, obere oder untere Dreiecksmatrix oder Diagonalmatrix ihr Abspeichern in reduzierter Form. Eine Reihe von Standardfunktionen, z.B. zur Determinantenberechnung, Rangbestimmung oder Ermittlung extremaler Elemente ergänzen die Sprache.

Der mit DEPOT2a erzeugte Vorübersetzer übersetzt M-FORTRAN-Programme in FORTRAN/ESER-Programme, in denen zur Ausführung der Feldoperationen Assembler-Unterprogramme des zugehörigen Laufzeitsystems aufgerufen werden. Von diesen Laufzeitroutinen wird zur Verarbeitung von Feldern bzw. Teilfeldern der Matrixmodul ES1055/MAMO benutzt.

3. Die 3D-Modellierungssprache GEKO

GEKO (Geometrische Konstruktion) ist eine Fachsprache zum gleichnamigen 3D-Modellierungssystem (siehe /6/). Sie ermöglicht die Beschreibung geometrischer Objekte und Operationen.

Zu den Objekten gehören elementare konvexe und zusammengesetzte. Die elementaren Objekte werden mit Definitionsanweisungen folgender Gestalt beschrieben: DEF a = s x;
Hiermit wird die Variable a als geometrisches Objekt vom Typ s definiert, das den durch x beschriebenen Wert erhält.
Für s kann eines der Schlüsselworte

PUNKT, GERADE, EBENE,
STRZ (Streckenzug),
RECHTECK, DREIECK, KREIS,
QUADER, PYRAMIDE, PSTUMPF, ZYLINDER, KEGEL, KSTUMPF,
POLY (konvexe Hülle) stehen.

Die Beschreibung x besteht z.B. aus

drei Komponenten (PUNKT),
einer Punktliste (STRZ),
Länge, Breite und Höhe (QUADER) oder
einer Objektliste (POLY).

Die wesentlichen Operationen mit geometrischen Objekten sind Verknüpfungen und Transformationen.
Verknüpfungen:

A+B Vereinigung (bzw. Zusammensetzen) der Objekte A und B,
A-B Differenz, als Resultat entsteht der Teil des Objektes A, der nicht in B enthalten ist,
A∩B Durchschnitt von A und B.

Beispiel: VOP E = A-B∩(C+D);

Transformationen:

VERSCH E VEKTOR F;
Verschieben des Objektes E um den Vektor F.

DREH E GERADE G WINKEL W;
Drehen des Objektes E an der Achse G um den Winkel W.

Weiterhin können Objekte kopiert, gestrichen und elementare in zusammengesetzten ersetzt werden.

Konstruierte geometrische Objekte lassen sich in 3D-Form (alle internen Daten) ein- und ausgeben, was zur Arbeit mit Modellkatalogen oder zum Anschluß anderer Systeme, z.B. zu statischen Berechnungen, genutzt werden kann. Weiterhin ist die Ein- und Ausgabe in 2D-Form (projeziert) zur bildlichen Darstellung möglich. Hierzu können Projektionsart ausgewählt sowie Angaben über die Darstellung verdeckter Kanten und Mantellinien angefügt werden.
Die Fachsprache wird ergänzt durch Rechenoperationen mit ganzen und reellen Zahlen, einfache Elemente der Vektorrechnung, bedingte Anweisungen und Laufanweisungen sowie Unterprogrammtechnik.
In einer Systembibliothek wird eine Reihe von Standardunterprogrammen bereitgestellt.
Da GEKO geometrisch allgemein gehalten und mathematisch gut abgesichert ist, bringt ihr Einsatz in verschiedenen CAD-Systemen Gewinn, z.B. im Maschinenbau oder Bauwesen.

Neben einem Interpreter für Dialogbetrieb wurde mit Hilfe von DEPOT2a ein Vorübersetzer realisiert, der GEKO-Programme in FORTRAN übersetzt.
Eine GEKO-Zylinderdefinition einschließlich ihrer Übersetzung kann z.B. mit folgender ML2a-Prozedur beschrieben werden:

```
ZYL: <DEF> NAME <=> <ZYLINDER>  !!
     <RADIUS> EXPR[1]  <HOEHE> EXPR[2]
->   NEXT RECORD  <KSTEU=504>
     NEXT RECORD  <IPARA(1)=> NAME_
     NEXT RECORD  <RPARA(1)=> EXPR_[1]
     NEXT RECORD  <RPARA(2)=> EXPR_[2]
   ON ERROR( PRINT('DEF Z=ZYLINDER RADIUS R HOEHE H')
             ERR )
```

Von dieser Prozedur werden die ML2a-Prozeduren NAME (Variablenname), EXPR (arithmetischer Ausdruck) und ERR (Fehlerstabilisierung) aufgerufen sowie Mit NAME_ und EXPR_ die durch diese Prozeduren erzeugten Zieltexte weiterverarbeitet.
Der erzeugte FORTRAN-Text besteht aus einer Reihe von Belegungen globaler Parameterfelder, die dann vom Laufzeitsystem bei Abarbeitung der Aktion abgefragt werden.

4. Datenprüfen mit DAPRUE

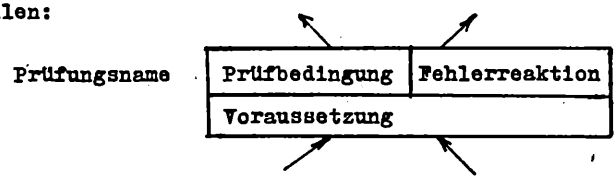
Überall, wo große Datenmengen anfallen, muß man mit dem Problem der Fehler bei der Datenerfassung fertigwerden. Üblicherweise versucht man, durch ein vorgeschaltetes Datenprüfen einen möglichst großen Prozentsatz dieser Fehler vor der eigentlichen Verarbeitung zu finden. Auch für das Datenprüfen ist die Verwendung einer geeigneten Fachsprache erheblich effektiver als etwa das Programmieren der entsprechenden Prüfprogramme in PL/I oder gar Assemblersprache. Auch Entscheidungstabellentechniken sind noch zu allgemein und wenig angepaßt.
Mit DAPRUE (siehe /2/) können Daten, die eine feste Recordstruktur haben, geprüft werden. Auch das Einteilen einer zu prüfenden Datenmenge in Gruppen, wobei jede Gruppe eine andere Struktur aufweist, ist möglich. Für die Prüfungen sind logische Ausdrücke, Vergleichsrelationen und die Mengen-Element-Relation zugelassen. Die einzelnen Felder der Records können wahlweise als ganze Zahlen oder Zeichenketten interpretiert werden.

Anhand eines Beispieles soll das Arbeiten mit DAPRUE erläutert werden. Gegeben sei eine Lochkartendatei, deren Sätze folgenden Aufbau haben:

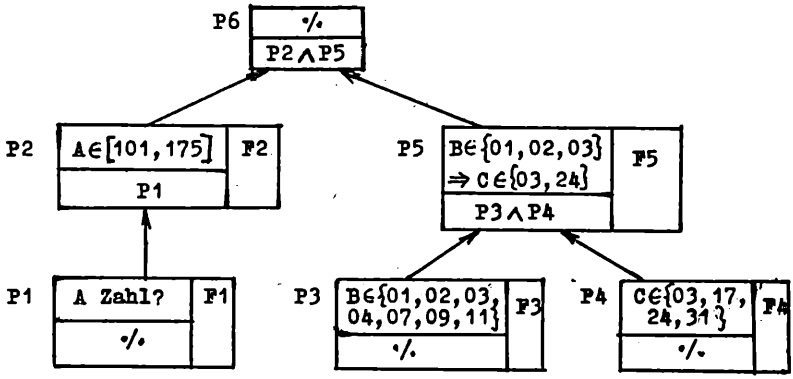
Merkmal	Spalten
A	1-3
B	4-5
C	6-7

An die Datensätze werden folgende Prüfbedingungen gestellt:
1. A ist eine Zahl aus dem Intervall $[101, 175]$.
2. B ist ein Element der Menge $\{01, 02, 04, 07, 09, 11\}$.
3. C ist ein Element der Menge $\{03, 17, 24, 31\}$.
4. Wenn gilt $B \in \{01,02,03\}$, dann soll gelten $C \in \{03,24\}$.

Diese Prüfungen in ihren gegenseitigen Abhängigkeiten lassen sich sehr gut in einem Graphen mit folgenden Kanten und Knoten darstellen:



Für das Beispiel ergibt sich der Graph (in diesem Fall ein Baum):



Das Erstellen eines solchen Graphen gehört zur Technologie, die mit DAPRUE mitgeliefert wird. Ein DAPRUE-Programm stellt einen derartigen Graphen in linearisierter Form dar.
Zunächst werden nach dem Schlüsselwort STRUCT die Felder des Records Bezeichnet und mit Längen versehen. Dann wird hinter SCHEMA die Startprüfung, hier P6, notiert. Anschließend werden sämtliche Prüfungen in beliebiger Reihenfolge aufgeschrieben. Eine Prüfung hat die Gestalt:

prüfungsname: ASSUME voraussetzung CHECK prüfbedingung
ERROR fehlerreaktion ;

Eine Fehlerreaktion Besteht aus einer Nummer, für die im anschließenden DEKO-Teil eine Fehlernachricht angegeben wird, und aus den Bezeichnungen der falschen Positionen.

DAPRUE-Programm zum obigen Beispiel:

```
PROG BEISPIEL
STRUCT A(3),B(2),C(2),X(73);
SCHEMA  P6: ASSUME P2&P5;
P1: CHECK ZAHL(A)  ERROR 1,A;
P2: ASSUME P1  CHECK A>=101 & A<=175  ERROR 2,A;
P3: CHECK B<:[01,02,03,04,07,09,11]   ERROR 3,B;
P4: CHECK C<:[03,17,24,31]            ERROR 4,C;
P5: ASSUME P3&P4  CHECK B<:[01,02,03] & C<:[03,24]
                  |¬(B<:[01,02,03])   ERROR 5,B,C;
DEKO 1, 'unzulaessiges alphazeichen';
     2, 'falsch: A nicht im intervall [101,175]';
     3, 'falsch: B laut nomenklatur';
     4, 'falsch: C laut nomenklatur';
     5, 'folgebeziehung verletzt: B und C';
END;
```

DAPRUE wurde am Rechenzentrum der APW entwickelt, mit DEPOT realisiert, d.h. ein Vorübersetzer in PL/I hergestellt, und seit mehreren Jahren mit Erfolg angewendet. Damit gelang es, den Projektierungsaufwand für die Realisierung von Datenprüfaufgaben auf weniger als ein Viertel des bis dahin üblichen Aufwandes zu senken.

Literatur:

/1/ Bormann,J./ Lötzsch,J.: Definition und Realisierung von Fachsprachen mit DEPOT, Diss., TU Dresden 1974

/2/ Drüge,W./ Nager,K.: Das Prüfschema als Beitrag zur rationellen Projektierung, RT/DV Heft 10/1983, S.30-33

/3/ Großmann,R./ Lötzsch,J./ Nager,K.: Automatisierte Erzeugung von Software für die Analyse und Verarbeitung von Fachsprachen (Metasystem DEPOT2a), Schriftenreihe des WBZ MKR/IV der TU Dresden, Heft 71/84

/4/ Großmann,R./ Hutschenreiter,J./ Lampe,J./ Lötzsch,J./ Mager,K.: DEPOT2a - Metasystem für die Analyse und Verarbeitung verbundener Fachsprachen (Anwenderhandbuch), TU Dresden, Sekt.Mathematik 1984 (im Druck)

/5/ Großmann,R./ Spantig,J.-U.: Die effektive Unterstützung der Arbeit mit dem Matrixmodul durch Entwicklung geeigneter fachsprachlicher Mittel und ihrer Implementation mit Hilfe des Fachsprachensystems DEPOT2a/ESER, Manuskript, TU Dresden, Sekt.Mathematik 1984

/6/ Hartwig,A./ Hernadi,A./ Mager,K.: GEKO - geometrische Grundsoftware mit Fachsprache zur Darstellung und Manipulation von Objekten im dreidimensionalen Raum, Schriftenreihe des WBZ MKR/IV der TU Dresden, Heft 66/83, S.24-33

/7/ Hutschenreiter,J.: Grundlagen und Implementation von verbundenen Fachsprachen, Vortragsmanuskript zur Tagung bit'85

TU Dresden, Sekt.Mathematik
8027 Dresden, Mommsenstr.13

Mager,K.

R ü d i g e r , Peter

R A E X I - Vorstufe eines Expertensystems für Kleinrechner

Bei der Entwicklung von RAEX I wurde von folgenden Ideen ausgegangen:

- Bei der Speicherung von Wissen ist es notwendig, Strukturen und Prozesse bzw. Arbeitsprozesse abzubilden.

- Wissen kann in verschiedenen Komplexitäts- oder Verallgemeinerungsniveaus gespeichert werden.

- Objektiv und subjektiv ist ein Prozeß eine Zustandsfolge.

- Bei der Repräsentation der Zustandsfolge im Gedächtnis existiert ein Zustand in seiner verbalen Beschreibung, und ihm sind seine Voraussetzungen und Folgen, die Möglichkeiten, ihn durch Aktionen zu verändern, und Zielzustände zugeordnet.

- Bei notwendiger differenzierterer Nutzung des Gedächtnisbesitzes realisiert das Gehirn eine differenziertere Abbildung der Realität über die Eigenschaften der Zustände.

- Eigenschaften existieren im Gedächtnis in ihrer verbalen Beschreibung in Verbindung mit einer Quantität.

- Der Vergleich zwischen zwei Zuständen im Gehirn kann nur bei formaler Gleichheit der Abbildungen der Zustände realisiert werden.

Die zwei folgenden Abbildungen widerspiegeln den beschriebenen Ansatz:

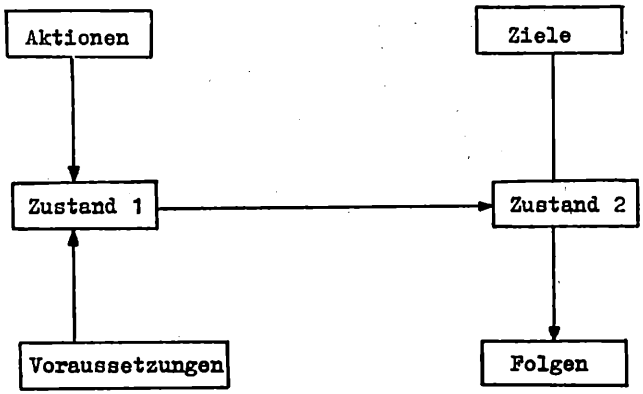


Abb. 1 Modell für die Abbildung eines Arbeitsprozesses im Gedächtnis

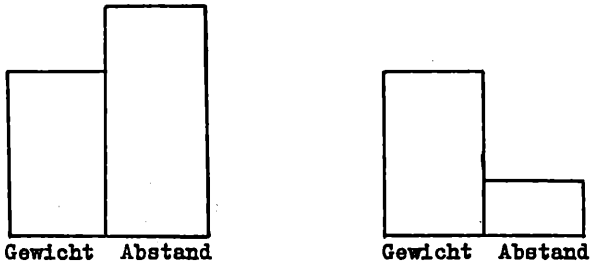


Abb. 2 Vergleich zweier Zustände im Gedächtnis

In den Abbildungen nicht enthalten ist die Tatsache, daß ein Problem erkannt und in dem existierenden Wissens- bzw. Erfahrungsschatz zugeordnet werden muß.

Im Gehirn wird diese Zuordnung über von den Psychologen sobenannte semantische Netze realisiert.

Wir haben jetzt einen Ansatz formuliert und wollen Ihnen nun mit der Erläuterung der Möglichkeiten von RAEX 1 zeigen, wie der Ansatz umgesetzt wurde.

Sie haben vor sich einen leeren Wissensspeicher, den Sie völlig frei mit Begriffen füllen können, die Sie nur von Ihrem ganz persönlichen Bedarf notwendiger Informationen abgeleitet haben, und die für Sie eine zentrale Bedeutung besitzen.
Stellen wir uns die Situation nach diesem ersten Schritt bei einem Abteilungsleiter software-entwicklung vor.

ARBEITSGRUNDLAGE	ARBEITSZIELE	BASISPROZESSE
PROGR.SPRACHEN	LITERATUR	ARBEITSPARTNER
EXPERTENSYSTEME	KNOWLEDGE ING	KADER
GESETZL.GRUNDL.	PROJEKTE	

Der zweite Schritt besteht in der Untergliederung jedes einzelnen Begriffs in Unterbegriffe. Im ersten Schritt haben wir eine erste Ebene gefüllt, und wir ordnen im zweiten Schritt jedem Begriff der ersten Ebene Begriffe der zweiten Ebene zu. Das geschieht so, daß wir uns einen Begriff der ersten Ebene auswählen, z. B. ' LITERATUR ' und für diesen Begriff die zweite Ebene füllen; wir füllen entweder mit Klassennamen für die Literatur oder schon mit Verfassernamen.

Bei RAEX I können Sie eine Begriffshierarchie bis in die sechste Ebene realisieren. Da in diesem Teil des Projekts nur die Begriffe behandelt werden, sprechen wir von der Arbeit im Begriffsraum.

Im Begriffsraum können Begriffe gespeichert und gelöscht werden.

Das folgende Bild zeigt einen Ausschnitt eines Begriffsraumes, so wie er nach einigen Wochen entstanden sein könnte.

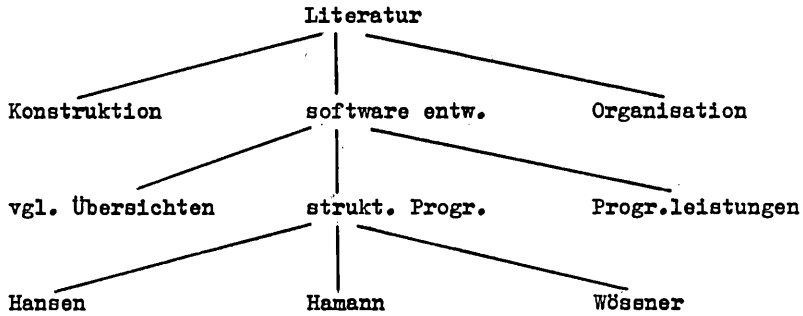


Abb. 3 Beispielausschnitt eines Begriffsraumes

Wir haben uns jetzt einen Überbau, eine Strukturierung bzw. Klassifizierung unseres Wissensbesitzes für die Zwecke unserer Arbeit geschaffen. Als Beispiel wurde individuelle Strukturinformation verwendet. Vergessen wir nicht, es gibt auch Strukturinformationen, die in der Gesellschaft vereinbart sind, und es gibt auch Prozeßinformationen. Für die Erläuterung des Prinzips hat diese Unterscheidung keine Bedeutung, wohl aber für die Benennung und für das Verständnis des nun folgenden Raumes, dem Raum der sachlichen Informationen zum übergeordneten Begriff bzw. dem Raum der Zustandsbeschreibungen.
Dieser Raum kann von jeder Ebene erreicht werden. In ihm sind die Definitionen von Zustandsfolgen oder von Strukturelementen in 2 x 80 Zeichen gespeichert. Unser Beispiel von Abb. 3 fortführend stellen wir uns hier einen Buchtitel vor:

Einführung in das Programmieren in LISP
Verlag Walter de Gruyter, Berlin (West) 1982

Ein Beispiel für die Speicherung eines Zustandes ist:
'Das Galtonbrett steht senkrecht, eine Kugel befindet sich im Startloch.'

In einer konkreten Anwendung des Systems sind in diesem Raum zum Zustand oder Strukturelement vereinbarte Zusatzinformationen speicherbar. Bei der Abbildung von Prozessen kann das z. B. eine Aktion sein. Verwendet man das System zur Terminkontrolle, würde der Text eine Aufgabe sein, und als feste Zusatzinformation sind z. B. Auftraggeber und Endtermin sinnvoll.
Die Gestaltung dieses Raumes wird am stärksten durch die Wünsche und speziellen Forderungen des Nutzers geprägt. Hier beschreibt der Anwender in verschiedenen Differenzierungs- bzw. Hierarchieniveaus seine Prozesse oder Strukturelemente so, wie es in der Kommunikation üblich ist. Für Prozessinformationen müßte in diesem Raum ein Zustandsnetz realisiert sein; die Anwendung von RAEX I machte dies nicht erforderlich, hier existieren einfache Zustandsketten.

Sollte diese Beschreibung nicht hinreichend genau oder sollten Ergänzungen notwendig sein, kann man diese zum Zustand oder zur sachlichen Information im Eigenschaftsraum speichern. In diesem Raum kann man einen Eigenschaftsnamen und einen 6 x 80 Zeichen langen Text zur Beschreibung der Eigenschaft speichern. Dieser Raum enthält also eine differenzierte Beschreibung des Zustandes, den Versuch der Beschreibung des internen Zustandsabbildes.
Die interne Repräsentation von Zuständen ist das Fundament für Denken und Kommunikation, der Eigenschaftsraum ist das Fundament für die Anwendung des Systems.
Hier können Verweise auf Abbildungen in Büchern, Karteien, Hinweise zur Realisierung des Zustandes, zu seinen Voraussetzungen und Folgen gespeichert werden.

Voraussetzung zur Beherrschung dieses Raumes, zur Beherrschung dieses Systems, verehrte Damen und Herren, ist die Ordnung unseres Wissens, ist die Beherrschung der Arbeitsmittel, ist die Überlegenheit des Geistes über die Basisprozesse. Auch deshalb heißt so ein System Expertensystem. Voraussetzung für die Anwendung eines Expertensystems ist Klarheit über den abzubildenden Problemraum oder Wissensbereich.

Wir haben das System bis jetzt von der Seite der Eingabe kennengelernt. Wenden wir uns der Seite der Auswertungen zu.

RAEX I wurde als Terminkontrollsystem entwickelt. Das bedeutet, der Raum der sachlichen Informationen heißt Aufgabenraum. Das bedeutet aber auch einen bemerkenswerten Unterschied für das Auswertungsverhalten des Systems. Bei der Terminkontrolle sollen in einer wichtigen Auswertungsvariante alle Aufgaben eines vorgegebenen Zeitraumes angezeigt werden - also unabhängig vom direkt übergeordneten oder von den übergeordneten Begriffen. Dagegen soll bei Prozeß- und Strukturinformation nur die Kammer des Raumes der Zustandsbeschreibungen bzw. der sachlichen Informationen angezeigt werden, die dem übergeordneten Begriff direkt zugeordnet ist.

Wenn ich bei Terminkontrolle die Tagesaufgaben zu sehen wünsche, möchte ich natürlich alle der Forderung entsprechenden Aufgaben sehen, ohne erst die Begriffe abklappern zu müssen. Wenn ich das System als Wissensspeicher nutze, interessiert mich niemals das ganze gespeicherte Wissen, sondern Wissen eines konkreten Bereiches auf einem konkreten Differenzierungsniveau.
Dieser Unterschied differenziert Systeme zur Wissensaufbewahrung und -vermittlung und Systeme für die Leitung von Entwicklungs- und Forschungsprozessen.

RAEX I wurde für die Leitung eines Entwicklungsprozesses geschaffen. Die Unterstützung bei der Realisierung der Aktenordnung war eine Zusatzforderung. Ein Abbild der Aktenordnung ist aber Wissen, die Abbildung in die Speicherstruktur Wissensspeicherung. Das heißt, beide sich grundsätzlich unterscheidenden Auswertungen mußten dem Anwender zur Verfügung gestellt werden.

Für die Auswahl von Aufgaben existieren in RAEX I folgende Möglichkeiten:

- zu einem Begriff einer Ebene
- ohne Begriffszuordnung

- zu einem Auftragnehmer
- zu allen Auftragnehmern

- die erfüllten Aufgaben
- die nicht erfüllten bzw. verlängerten Aufgaben

- Aufgaben in einem Zeitraum
- Tagesaufgaben

- Aufgaben nach ihrer Wichtigkeit sortiert

Die beschriebenen Eingabe- und Ausgabemöglichkeiten lassen dem Nutzer des Systems einen großen Spielraum; er kann das System den Notwendigkeiten seiner Arbeit und seinen Bedürfnissen anpassen. Diese Variabilität ist aber auch eine Gefahr. Die Gefahr heißt Unordnung in der Vorstellung über die Basisprozesse. Das System setzt Ordnung im Problemraum und Klarheit in der Vorstellung des Nutzers voraus. Diese Voraussetzung ist eine Forderung für viele komplexe Arbeitsprozesse und erhält durch dieses Projekt einen objektiven Maßstab, an dem sich der Nutzer messen kann.

Wir haben bisher den theoretischen Ansatz und den Kern der Realisierung erläutert. Im weiteren soll klar werden, wie sich RAEX I in das Leitungssystem einordnet und welcher Nutzerkreis möglich ist.

Wenn man das System am Bildschirm aufruft, erscheint die Ausschrift: IHRE PERSONALNUMMER, BITTE:

Sie geben Ihre Personalnummer ein, und es wird geprüft, ob die Nummer in der Datei der zugelassenen Nutzer enthalten ist. Wenn sie enthalten ist, erscheint die Ausschrift: DAS PASSWORT, BITTE: Das Passwort wird ebenfalls auf Zulässigkeit geprüft. Ist es unzulässig, wird das Programm beendet.

Das Passwort trennt die Informationsbestände für verschiedene Nutzer oder Informationsbestände eines Nutzers. Der zweite Fall ist sinnvoll, wenn sich eine Person mit sehr unterschiedlichen Arbeitsgebieten beschäftigt.
Beliebig viele Nutzer können also mit ihrem eigenen oder bzw. und mit einem gemeinschaftlichen Datenbestand arbeiten.

Hat das System das Passwort akzeptiert, erscheint eine Funktionsauswahl:

BEGRIFFSRAUM	1
SCHLUESSEL	2
POST	3
ENDE	Ø

Wählt man die 1, so kann man mit dem System als Terminkontrollsystem arbeiten. Man kann verschiedene Begriffs- und Aufgabenbereiche auswählen, Begriffe speichern und löschen, Aufgaben speichern, modifizieren - z. B. Terminverlängerungen oder Erfüllungsmeldungen eingeben -, löschen. Und man kann Ergänzungen zu den Aufgaben speichern oder sich anzeigen lassen.

Eine wichtige Rolle bei der Terminkontrolle spielen die Arbeitspläne. Da die Aufgaben gespeichert sind, liegt es nahe, sich die Arbeitspläne von RAEX I drucken zu lassen. Auf dem so erzeugten Arbeitsplan stehen die Aufgaben, die Endtermine, die Quelle der Aufgabe, das Kennzeichen für den Bearbeitungsstand und eine Schlüsselnummer für die Aufgabe. Der Arbeitsplan ist nach Auftragnehmer sortiert. Will der Nutzer nur eine spezielle Aufgabe sehen, erhält er mit der Wahl der Funktion 'SCHLUESSEL 2' die Möglichkeit, die Schlüsselnummer einzugeben und erreicht damit, daß das System die gewählte Aufgabe anzeigt.
Wählt man die Funktion 'POST 3', kann man eine Nachricht bzw. Aufgabe oder Anfrage an ein anderes Bildschirmgerät senden oder Nachrichten von anderen Bildschirmgeräten empfangen.

Damit ist das System zu einem einfachen Kommunikationsmittel geworden. Es spart Papier und Zeit und vermindert die Rolle subjektiver Fehler bei der Informationsübermittlung.
Die Daten, die auf diese Weise ausgetauscht werden, können noch nicht einfach im eigenen Begriffsraum zugeordnet werden. Ob das notwendig ist, wird die Erfahrung zeigen. Bis jetzt bleiben die Daten auf dem 'Postamt' und können dort separat behandelt werden.
Ein besonderer Service besteht in der Behandlung von periodisch wiederkehrenden Terminen. Diese müssen nur am Beginn der Arbeit mit dem System eingegeben werden.

Anhand einer angegebenen Zeitspanne erzeugt das System später die Termine für Leitungs-, Dienst- und Arbeitsberatungen selbst.

Wir haben jetzt das System RAEX I in seinen wesentlichen Eigenschaften beschrieben. Wir haben gesehen, daß es in der Lage ist, Wissen unterschiedlicher Art zu speichern und in einer Ordnung zu verwalten. Stellen wir uns nun die Frage, unter welchen Bedingungen sich so ein System als nützlich erweist. Unter welchen Bedingungen wird so ein System effektiv?

So ein System ist nützlich, wenn

- die Basisprozesse eine hohe Komplexität aufweisen,
- Einzelpersonen diese Komplexität beherrschen müßten, um ihre Arbeit in hoher Qualität und Quantität zu realisieren,
- relativ lange Einarbeitungszeiten in Arbeitsgebiete existieren,
- einige Aufgaben selten vorkommen, aber ein komplexes Herangehen an die Lösung erfordern,
- sich ein komplexer Informationsraum häufig verändert und mehrere Personen mit dem aktuellen Stand arbeiten müssen, die Zeit für Informationsaustausch aber fehlt,
- es auf hohe Genauigkeit in der Informationsübermittlung ankommt (die Genauigkeit muß in der Formulierung durch den Eingebenden erzeugt werden - das System stimuliert die redundanzarme Eingabe).

Wir wollen jetzt an zwei einfachen Beispielen die Arbeit mit RAEX I demonstrieren. In Beispiel 1 beschreiben wir einen Handelsprozeß und speichern für ein Arbeitsgebiet die gesetzlichen Grundlagen. Unter dem Begriff 'HANDEL' seien gespeichert 'EXPORT', 'IMPORT', 'BINNEN'. Und unter 'EXPORT' seien gespeichert 'ANGEBOTSPROZESS', 'VERTRAGSPROZESS', 'REALISIERUNG'.
Im Begriffsraum sei die Begriffsfolge

'HANDEL' 'EXPORT' 'VERTRAGSPROZESS'

ausgewählt worden, die Ebene 4 sei leer.
Wir speichern zu Ebene 4 den neuen Begriff

'GESETZLICHE GRUNDLAGEN'

und gehen in den Aufgabenraum über.

Der Bildschirm zeigt jetzt das Bild:

```
                                        AUFGABENRAUM
HANDEL              EXPORT              VERTRAGSPROZESS
REALISIERUNG

                : AE
```

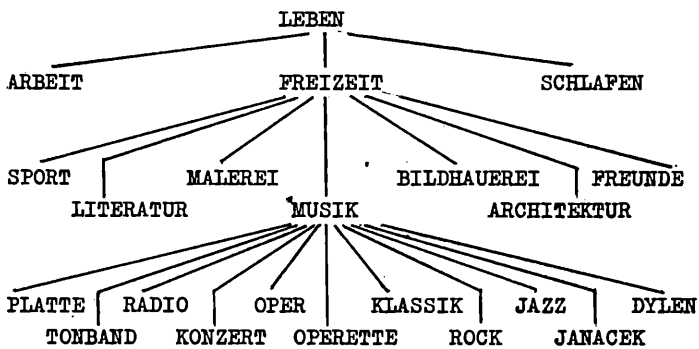
Mit den Buchstaben AE melden wir eine Eingabe an, und wir können an der mit '-' bezeichneten Stelle eine gesetzliche Grundlage eingeben, z. B.

'VTG 1197'

'MINISTERRAT' '12.11.1984'

Dieses Beispiel zeigt ein Problem. Für jeden Begriff der zweiten Ebene gibt es gesetzliche Grundlagen, d. h., der Begriff muß mehrfach gespeichert werden. In einer Systementwicklung bietet sich für dieses Problem eine Rationalisierung an.

Das zweite Beispiel soll unseren Lebensprozeß beschreiben. Wir fragen das System nach den Möglichkeiten, unsere Freizeit zu gestalten. Im Begriffsraum sei gespeichert:



Im Aufgabenraum können hier die Möglichkeiten der Freitzeitgestaltung zu den konkreten Begriffen gespeichert sein.

An diesem Beispiel werden zwei weitere Probleme deutlich.
In Ebene 4 sehen wir, daß Begriffe der Tonerzeugung nach ihrer physikalischen Herkunft differenziert neben Begriffen der Genrezugehörigkeit von Musikstücken und Begriffen, welche Komponisten und Sänger bezeichnen, existieren. Für einen Wissensspeicher für den individuellen Gebrauch ist das sicher kein Problem. In Wissensspeichern für die Gemeinschaft muß jedoch sorgfältig erwogen werden, in welchen Fällen man besser eine weitere Zwischenebene schafft.
Das zweite Problem existiert ebenfalls für eine gemeinschaftliche Anwendung des Systems. Wenn eine Stelle die Konzertpläne eingeben würde und diese für alle Nutzer zur Verfügung ständen, wäre doch ein gutes Stück gesellschaftlichen Arbeitsaufwandes gespart, und die Systemnutzer würden beim aktuellen Bedürfnis nach einem DYLEN-Konzert angezeigt bekommen, daß, wo und wann so ein Konzert stattfindet.
Die Auswertung Veranstaltungen eines Zeitabschnitts muß selbstverständlich auch zur Verfügung stehen.

Nun ist folgende Frage interessant: Wie kann das System zu einem Expertensystem werden?
Zunächst ist die Antwort einfach: indem ein Experte für Leitungstätigkeit oder eine Gruppe von Experten für den geforderten Arbeitsbereich einen Anfangsdatenbestand schafft, d. h. das Fachwissen und das Leitungswissen speichert, die Beratungsperioden und Standardabläufe festlegt. Wer sind denn aber die Experten für ein Wissensgebiet, und wo sind die Grenzen für ein Wissensgebiet? Welche Wissensgebiete enthalten aggregiertes Wissen oder Wissen über analoge bzw. vergleichbare Prozesse, welche Regeln gibt es zur Verallgemeinerung von Wissen, ist Verallgemeinerung überhaupt sinnvoll? Das sind Fragen, auf die die Praxis eine Antwort geben muß.
Wer sich für so ein System entscheidet, sollte sich wenigstens drei Fragen gestellt und entsprechend drei klare Antworten formuliert haben.

Die Fragen sind:

- Für welches Wissensgebiet soll das System genutzt werden?
- Wie soll sich das System in die bereits existierende Organisation einfügen?
- Wodurch ergibt sich der zu erwartende Nutzen?

Verantwortungsbewußte Antworten sind der Garant für die Erfüllung der Erwartungen.

Werfen wir nun noch einen kurzen Blick in die Zukunft.

Für das System lassen sich zwei wesentliche Entwicklungsrichtungen erkennen. Die erste Richtung enthält die Analogien zu den CAD-systemen, z. B. ein System zur Unterstützung der software-entwicklung oder ein System zur Unterstützung der Erfindungstätigkeit. Bei diesen Anwendungsfällen liegt das Problem eher bei der Schaffung geeigneter Wissensfonds - sie sind also prinzipiell realisierbar.
Die zweite Richtung zielt auf die Wissensverarbeitung. Für diese Richtung ist erstens noch Grundlagenforschung der Psychologie über die Dialektik und Effektivität von anschaulichem und begrifflichem Denken bei Problemlösungsprozessen notwendig, zweitens scheint es notwendig, daß die hardware-hersteller begriffsverarbeitende Bauelemente produzieren bzw. solche Bauelemente produzieren, die die Begriffsverarbeitung optimal unterstützen.

Diese Forderung sei kurz erläutert:

Jeder Zustand läßt sich in der Form von Abb. 2 darstellen.
Im Computer ist die Darstellung

ZN: E1Q1M1 E2Q2M2 ....

ENQNMN mit
ZN = Zustandsname
E = Eigenschaftsname
Q = Quantität
M = Maßeinheitname möglich

Für den Computer sind Namen gleich Adressen; an diesen Adressen sind Zeichenketten gespeichert. Der Mensch versteht diese Darstellung in ihrer externen Form auch:

Raumklima: Zugluft 2 m/s
Temperatur 20° C

Die Reaktion auf Zustände hängt von den Zielstellungen der Menschen ab; diese Ziele sind aber auch Zustände und durch Aktionen zu erreichen. Die Beschreibung der Aktionen ist speicherbar.

Diese Darstellung erfordert für ihre Verarbeitung außerhalb des Gehirns geringen Aufwand und ist für alle Vorgänge des Denkens im rationalen Bereich anwendbar. Sie bietet die Möglichkeit, einen Algorithmus für die Abbildung von Denken im Computer zu schaffen.

Notwendig für Wissensverarbeitung scheint weiterhin ein Bildschirm mit den Ausmaßen des Gesichtsfeldes des Menschen und ein frei gestaltbarer Zeichenvorrat.

Das Wesentliche an der Wissensverarbeitung, die direkte Unterstützung des menschlichen Denkens, läßt sich auch mit heute und hier existierender Technik realisieren. Beide Wege, die Entwicklung neuer hardware und die Entwicklung und Testung neuer software, müssen parallel beschritten werden.

Merken Sie sich bitte von diesem Vortrag das Folgende:

In der DDR existiert mindestens ein System, daß sich als Expertensystem eignet. Es ist auf vorhandener Technik implementierbar. Es ist ein Basissystem und an verschiedene Nutzerforderungen gut anpaßbar.

Expertensysteme können die Effektivität, die Qualität und die Sicherheit von Entwicklungsarbeit wesentlich erhöhen. Bei dem immer größer werdenden Komplexitätsgrad der Entwicklungen gibt es zu Expertensystemen keine Alternativen. Expertensysteme sind also keine Wunschvorstellungen von träumenden softwareentwicklern, sondern entscheiden über die Mark, die morgen uns oder anderen gehören wird.

Ich hoffe, alle, die es angeht, haben verstanden.

Dr.-Ing. Alfred Schilling

Anwendung des Compilererstellungssystems "CWS"

Es soll Aufbau und Wirkungsweise des "Compiler Writing-Sytems" (CWS) im Überblick dargestellt werden. Dieses in PASCAL programmierte System ist am ORZ der KMU Leipzig implementiert und wird zur Zeit umfassend erprobt.
Das Ergebnis der Anwendung dieses Systems ist ein vollständiger Compiler in PASCAL, wobei der Ausgangspunkt eine LL(1)-Grammatik der Quellsprache in einer speziellen Darstellungsform ist.

Dr.-Ing. A. Schilling
Karl-Marx-Universität Leipzig, ORZ
7o1o Leipzig
Karl-Marx-Platz

Staudte, Rainer

PROLOG-Interpreter fuer Mikrorechner

Einige Vorteile und Probleme der Anwendung von PROLOG auf Mikrorechnern werden diskutiert und erste Ergebnisse der Arbeit mit einem experimentellen PROLOG-System vorgestellt.

1. Bessere Anwendungsbedingungen der logischen Programmierung

Die unmittelbar auf der Basis mathematischer Logik beruhenden Programmiersprachen, unter ihnen PROLOG, haben sich in den letzten Jahren rasch verbreitet. Ein nicht unwesentlicher Grund fuer diese Popularitaet sind Implementationen auf relativ kostenguenstigen Mikrorechnern, die eine dezentrale Nutzung dieser Sprachen gestatten und ihre Anwendung nicht mehr auf wenige leistungsfaehige Rechenzentren beschraenken.
Als Beispiel dieses Trends sei Micro-PROLOG von Logic Programming Associates genannt, das etwa 240 Resolutionen je Sekunde fuer Z80-Prozessoren (4 MHz) erreicht. Micro-PROLOG existiert neben dieser Z80-Version in CP/M auch fuer 8088/86 in MSDOS und fuer weitere Mikrorechner u. a. auch in UNIX /1,2/.

2. Spezielle Probleme der Implementation von PROLOG

PROLOG-Implementationen fuer Mikrorechner erfordern die Loesung spezieller Aufgaben, die sich aus Hardware und Software dieser Klasse von Rechnern ableiten. Einerseits ist das die relativ geringe Operationsgeschwindigkeit der Prozessoren. Da PROLOG vorwiegend interaktiv genutzt wird, ist diese Restriktion nicht nachteilig, wenn die Antwortzeit mit der menschlichen Reaktionszeit vergleichbar bleibt.
Problematischer ist die Frage der Verwaltung des meist sehr beschraenkten Hauptspeichers. "Der gesamte Komfort der Sprache, darunter ihre Vorzuege wie Strukturen, Listen, Matching und Backtracking, werden teuer bezahlt, wenn man die dafuer notwendigen Hauptspeicheranforderungen in Betracht zieht" /3/. Prinzipielle Wege zur Loesung dieser Aufgabe sind dynamische Speicherverwaltung mit virtueller Adressierung /4/ oder modulare Organisation des PROLOG-Programms und Ueberlagerung.
Auch wenn die Implementationsfragen vor dem Anwender verborgen bleiben, so darf man bei der Beurteilung eines PROLOG-Systems ihre Existenz nicht vergessen.

3. Ein portabler PROLOG-Interpreter

Ungeachtet der zuletzt genannten speziellen Probleme einer PROLOG-Anwendung fuer praxisrelevante Aufgaben sind Mikrorechner ein geeignetes Hilfsmittel zur Untersuchung typischer Fragen der logischen Programmierung. In diesem Fall ist die Einbettung des Interpreters in eine problemorientierte Programmiersprache mit Entwicklungssystem sinnvoll.

Neben bekannten Implementationen in PASCAL und PLZ fanden die Arbeiten zur Entwicklung eines abgeruesteten PROLOG-Interpreters in verschiedenen BASIC-Versionen fuer die Mikrorechner PC 8001 und HC-20 bisher wenig Beachtung /5,6/. Beweis fuer die hohe Portabilitaet diese Interpreters ist seine Implementation und Weiterentwicklung im SCPX /7/. Sie ist besonders geeignet fuer Neulinge in PROLOG.
Eine weitere Moeglichkeit, einem noch breiteren Nutzerkreis zur logischen Programmierung Zugang zu verschaffen, ist die Umsetzung dieser Version auf Lerncomputer. Hier machen sich natuerlich geringer Hauptspeicherplatz und fehlender BASIC-Compiler zur Zeit noch sehr nachteilig bemerkbar.

4. Zum Test eines PROLOG-Systems

Wesentliche Merkmale zur Characterisierung eines PROLOG-Systems sind Operationsgeschwindigkeit und maximale Anzahl gleichzeitig aktiver Klauseln. Nachfolgende Beispiele gehoeren zu den Erfahrungen, die der Autor bei seiner ersten Bekanntschaft mit einem PROLOG-System und dessen Testung sammelte /8/.
Eine grobe Aussage zum Zeitverhalten kann man bei der klassischen Aufgabe zur Positionierung von acht Damen auf einem Schachbrett erlangen, indem man die Aufgabe auf n Damen auf einem n x n - Brett erweitert. Verzichtet man auf konstruktive Vefahren und wendet die Methode der vollstaendigen Durchmusterung an, so erweist sich hier die kombinatorische Vielfalt als hinreichend. Bereits ab n=6 ist ein merkliches Anwachsen notwendiger Backtrackingoperationen zu verzeichnen.
Verfeinerte Testverfahren unterscheiden zwischen Backtrackgeschwindigkeit und Realisierung reiner logischer Interferenzen. Ein (nicht ganz typisches) PROLOG-Programm fuer letzteres ist in folgendem Fragment eines Ablaufprotokolls von /7/ enthalten:

```
: list
1 fac(0,1).
2 fac(A,B) :- sub(A,1,C),fac(C,D),mul(D,A,B).
PROLOG OK
?-fac(2,Ergebnis).
- 2- 2- 1- o
+++ one answer is +++
    2
PROLOG OK
```

Die Ziffern nach dem Aufruf zur Berechnung von 2! protokollieren aktivierte Klauseln. Am Zeilenende steht o fuer einen erfolgreichen Beweis, x kennzeichnet ein Backtracking.
Folgendes Programm deckt leicht evtl. Limitierungen bei der Anzahl gleichzeitig aktiver Klauseln:

```
: load
file name ?FIBONA
PROLOG OK
: list
1 fibo(0,1).
2 fibo(1,1).
3 fibo(A,B) :- sub(A,1,C),sub(C,1,D),
               fibo(C,E),fibo(D,F),add(E,F,B).
PROLOG OK
?-fibo(4,Resultat).
- 3- 3- 3- 2- 1- 2- 3- 2- 1- o
+++ one answer is +++
    5
PROLOG OK
3 fibo(A,B) :- sub(A,1,C),sub(C,1,D),
               fibo(D,F),add(F,F,G),add(G,D,H),sub(H,1,B).
PROLOG OK
?-fibo(4,Resultat).
- 3- 3- 1- o
+++ one answer is +++
    5
PROLOG OK
```

Berechnet wird jeweils das fuenfte Element der FIBONACCI-Folge. Dabei erfolgt im ersten Aufruf die Berechnung streng nach Definition. Nach entsprechender Korrektur der Klausel dokumentiert der zweite Aufruf die reduzierte Beweislaenge.

Literatur

/1/ CLARK, K. L. / MCCABE, F. G.
Micro-PROLOG - Programming in Logic
Prentice Hall Int., Englewood Cliffs e. a., 1984

/2/ KUTZLER, B.
Erfahrungen mit Micro-PROLOG von LPA
Computer-Aided Mathematical Problem Solving, Linz, 1983

/3/ DOCHEV,D. / DICHEV, CH. / MARKOV, Z. / AGRE, G.
Problems in the Implementation and Use of PROLOG for Personal Computers
Inst. of Industrial Cybernetics and Robotics, Sofia, Preprint

/4/ ANTONOV, A. / DEMIREV, G. / STOYCHEV, L.
PROLOG Interpreter with Distributed Data Base for a Personal Computer
Inst. for Mechanical and Electrical Engineering, Varna, Preprint

/5/ MAZUDA, H. / TANAKA, K.
Experimentelle Anwendung von Micro Prolog auf Personalcomputer
Bit(Japan), 14(1982), No 7, 840-853

/6/ KADZI, K.
Software fuer Mikrorechner PC 8001 und HC-20
Bit(Japan), 15(1983), No 9, 1036-1057

/7/ STAUDTE, R.
MicroPROLOG-Interpreter fuer SCPX 1526 (Benutzungshinweise)
Technische Hochschule Karl-Marx-Stadt, Sektion Informatik,
1985

/8/ МЕДЫНЦЕВ, А. и др.
ПРОЛОГ СМ ЭВМ
ЛЭТИ, Ленинград, каф. МО ЭВМ, 1984

Dr. rer. nat. Rainer Staudte
Technische Hochschule Karl-Marx-Stadt
Sektion Informatik
9010 Karl-Marx-Stadt
PSF 964

Vojislav Stojković
Institute of mathematics
Studentski trg 16
11000 Belgrade
Yugoslavia

EXPERT PROBLEM SOLVING USING IF-THEN RULES

If-then systems are the problem-solving systems based on matching simple rules to given problems. The purpose of this paper is to summarize what if-then systems are and to show how they can be implemented in LispKit Lisp language, which is a dialect of Lisp language. The expert knowledge can be represented as collections of if-then rules, which have the following form:

```
IF    <trigger fact 1 is true>
      <trigger fact 2 is true>
       ......................
THEN  <conclusion fact 1 is true>
      <conclusion fact 2 is true>
       ........................
```

Systems based on if-then rules can do convincing medical diagnosis, natural language understanding, planning and problem solving, sequence prediction, and so on.

There are many ways to represent rules. The following one enables easy access to the rule parts:

```
(RULE  <name>
       (IF  <trigger fact 1>
             ..............
            <trigger fact n> )
       (THEN  <conclusion fact 1>
               ................
              <conclusion fact n> ))
```

The facts can be represented as lists of atoms

The problem solver can do forward and backward chaining. A problem solver is doing forward chaining if it starts with a collection of facts and tries all available rules over and over, adding new facts as it goes, until no rule applies. A problem solver is doing backward chaining if it starts with an unsubstantiated hypothesis and tries to prove it. The strategy involves finding rules that demonstrate the hypothesis and then verifying the facts that enable the rule to work.

Dr. Klaus Wiederänders

"PASCAL für Mikrorechner"

An der Sektion Mathematik der Humboldt-Universität werden seit mehreren Jahren Compiler für PASCAL-Sprachen auf kleinen Mikrorechnern entwickelt. Als Zielrechner fungieren K1520-Rechner (MRES, BC5120, OEM-Rechner) mit den Betriebssystemen MEOS, SCP, M/OS, UDOS etc. und in Zukunft deren 16-Bit Nachfolger. Quellsprachen sind PASCAL/M, Concurrent Pascal, Modula-2 und PASCAL/M2, für die an der Humboldt-Universität Mikrorechnercompiler vorliegen.

Vorgestellt wird im Detail die Vertragsentwicklung für das Kombinat Robotron zur Installation von PASCAL/M2 für BC5120 unter SCP.

Dr. K. Wiederänders
Humboldt-Universität Berlin
Sektion Mathematik
Bereich Informationsverarbeitung
1086 Berlin
PSF 1297

SEKTION 6
Systemsoftware

Bell, J.-P.
Grafik, W.
Müller, P.

On-line-Nutzung der Externspeicher des K 1600 für mehrere Bürocomputer A 5120

1. Vorbemerkungen

Im rechentechnischen Labor der Sektion Mathematik der Humboldt-Universität zu Berlin werden mehrere Bürocomputer A 5120 sowohl zur Forschung als auch zur Ausbildung einer großen Zahl von Studenten eingesetzt. Eines der Hauptprobleme bei diesem Einsatzprofil ist die Speicherung von Daten, d.h. Quelltexten, ausführbaren Programmsystemen und nutzerspezifischen Daten sowohl für die tägliche Arbeit der Wissenschaftler und Studenten als auch für längere Zeit zur Archivierung der gewonnenen Forschungsergebnisse (Datenumfang einer Compilerentwicklung PASCAL z.B. über 5 M Byte). Die Bürocomputer A 5120 verfügen standardmäßig neben den Minidiskettenlaufwerken K 5600 mit maximal 200 K Byte Kapazität (Formatierung 1024 Byte pro Sektor) über keine externen Speichermedien. Kassettenmagnetbandgeräte und 1/2"-Magnetbandgeräte sind serienmäßig nicht verfügbar. Zur Archivierung einer Compilerentwicklung werden z.B. ca. 25 Minidisketten benötigt. Um trotzdem die oben genannten Aufgaben langfristig lösen zu können, ist es notwendig, weitere externe Speichermöglichkeiten für die Bürocomputer nutzbar zu machen, da die notwendige Zahl von Minidisketten, jährlich zusätzlich mehrere hundert Stück, nicht zur Verfügung stehen. Eine Lösungsmöglichkeit für uns bietet sich in der on-line-Nutzung der Externspeicher, Kassettenspeicher CM 5400 und Magnetbandgeräte CM 5300 des Rechners K 1600 an. Serienmäßig stehen für eine derartige Verbindung zwischen Bürocomputer A 5120 und K 1600 die Schnittstellen IFSS und V.24 mit einer maximalen Datentransferrate von 9,6 K Bit/s d.h. ca. 1 K Byte/s zur Verfügung. Dies ist sehr langsam, wenn man berücksichtigt, daß mit einem Minidiskettenlaufwerk Datentransferraten bis zu 10 K Byte/s praktisch erreichbar sind. Außerdem müßte von Seiten des K 1600 jeder Bürocomputer über einen gesonderten

Kanal einer Anschlußsteuereinheit bedient werden, was die Zahl der anschließbaren Bürocomputer stark verringern würde. Ein Multiplexer K 8523 für den K 1600 würde das Anschlußproblem zwar lösen, aber die Datenübertragungsrate wäre auch nicht höher. Deshalb wird nachfolgend beschriebene Lösung realisiert.

2. Technische Beschreibung

Die technische Basis für die Kopplung des Rechners K 1600 mit mehreren Bürocomputern A 5120 besteht aus einer optoelektronischen Kopplung der Bürocomputer und eines OEM-Rechners K 1520 untereinander und einer Kopplung des Rechners K 1600 und des OEM-Rechners auf der Grundlage einer SIF 1000 Schnittstelle mittels Anschlußsteuereinheiten ADA. Die topologische Struktur dieses lokalen Netzes ist ein Ring, der in Abbildung 1 zu sehen ist.

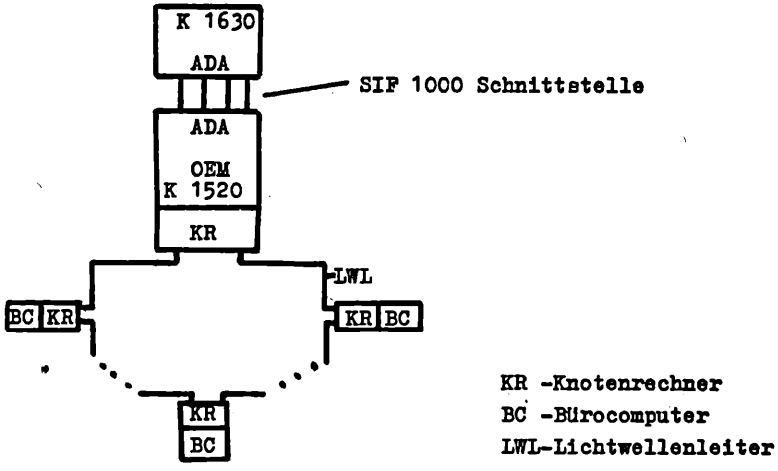


Abbildung 1: Struktur der Rechnerkopplung K 1600-Bürocomputer A 5120

Die optoelektronische Kopplung, die von der Sektion Elektronik der Humboldt-Universität zu Berlin entwickelt wurde,

arbeitet mit einer Datenübertragungsrate von 500 K Bit/s, d.h. ca. 60 K Byte/s. Die SIF 1000 Schnittstelle ermöglicht eine maximale Datenübertragungsrate von 20 K Byte/s. Damit ist zwischen den Rechnern K 1600 und A 5120 eine höhere Datenübertragungsrate möglich, als zwischen dem Prozessor des Bürocomputers A 5120 und einem angeschlossenem Minidiskettenlaufwerk. Beim Zugriff zu den externen Speichern des Rechners K 1600 werden also ähnliche Datenübertragungsraten wie beim lokalen Zugriff zu Minidiskettenlaufwerken erreicht.

Die von der Sektion Elektronik entwickelte optoelektronische Kopplung beruht auf einer optischen Informationsübertragung mittels Lichtwellenleiter. Dazu wird in jedem Bürocomputer und in dem OEM-Rechner zusätzlich ein Knotenrechner bestehend aus zwei Leiterkarten, Medienkarte und Buscontroler, integriert. Diese Leiterkarten haben ein Anschlußbild entsprechend des K 1520-Buses, so daß weitere Veränderungen an den Bürocomputern nicht notwendug sind. Die freien Steckplätze sind in der Regel vorhanden. Die Medienkarte realisiert die Umformung der optischen in elektrische Signale und umgekehrt. Gleichzeitig realisiert sie die Repeaterfunktion für das optische Medium falls die im Ring transportierten Daten nicht für den konkreten Knotenrechner sind oder falls der Bürocomputer nicht eingeschaltet ist (separate Stromversorgung der Repeaterfunktion). Der Buscontroler mit U880 Prozessor steuert den Transport der Daten zwischen Medienkarte und einem im Buscontroler lokalisierten RAM-Bereich, der als Dual-Port-Memory ausgebildet ist. Der Datenaustausch zwischen Knotenrechner und Bürocomputer wird mit Hilfe dieses Dual-Port-Memory realisiert, in dem dieser Speicherbereich zeitweise in den Adreßraum des Bürocomputers eingeblendet wird, wenn der Bürocomputer Daten empfangen bzw. senden will. Das Ein- und Ausblenden des Speicherbereichs wird durch den Bürocomputer gesteuert. Der Speicherbereich kann bis 8 K Byte groß sein. Der Knotenrechner meldet ankommende Datenblöcke durch Interruptsignale. Die Schnittstelle Knotenrechner-Bürocomputer entspricht bezüglich dem OSI-Referenze-Modell der Schicht 3 -Netzwerkschicht- mit

"Ende zu Ende" Kontrolle. Der Zugriff zum optischen Medium erfolgt durch den Knotenrechner mit dem Verfahren CSMA/CD. Die Abbildung 2 gibt einen Überblick über die Struktur der Kopplung von Bürocomputer und Knotenrechner.

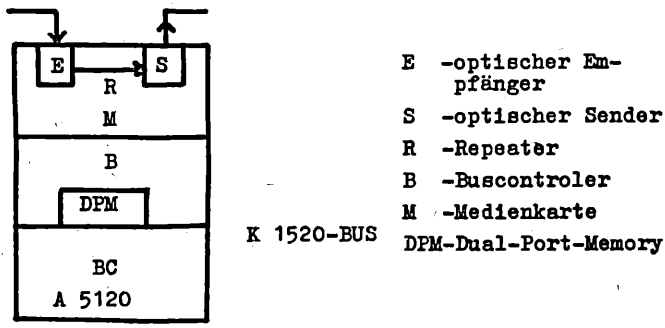


Abbildung 2: Struktur des Bürocomputers mit Knotenrechner

Bei der Kopplung K 1600-OEM K 1520 wird jeweils eine Anschlußsteuereinheit ADA mit Schnittstelle SIF 1000 benutzt, wobei die Anschlußsteuereinheit des OEM-Rechners leicht modifiziert wurde, da diese Schnittstelle eigentlich nicht für Rechner-Rechner-Kopplung konzipiert ist. Die genaue Kopplung der beiden Anschlußsteuereinheiten ist in Abbildung 3 dargestellt.

3. Struktur und Funktion der Koppelsoftware

Bei der Konzeption der Koppelsoftware wurden folgende Gesichtspunkte berücksichtigt:

1. Der Austausch von Datenpaketen zwischen beliebigen Rechnern des Ringes soll möglich sein, d.h. jeder Rechner des Ringes soll Dienste zur Verfügung stellen können und andererseits Dienste eines beliebigen anderen Rechners nutzen können.
2. Beim Ausfall eines Rechners soll trotzdem die Funktionsfähigkeit aller übrigen Rechner gesichert sein und deren Arbeitsfähigkeit nicht weiter durch den ausgefallenen

Rechner beeinflußt werden.

3. Die Schnittstellen der Koppelsoftware zum jeweiligen Betriebssystem sollen möglichst einfach und übersichtlich sein, um eine in Struktur und Funktion gleichartige Koppelsoftware für alle beteiligten Rechner erstellen könen.

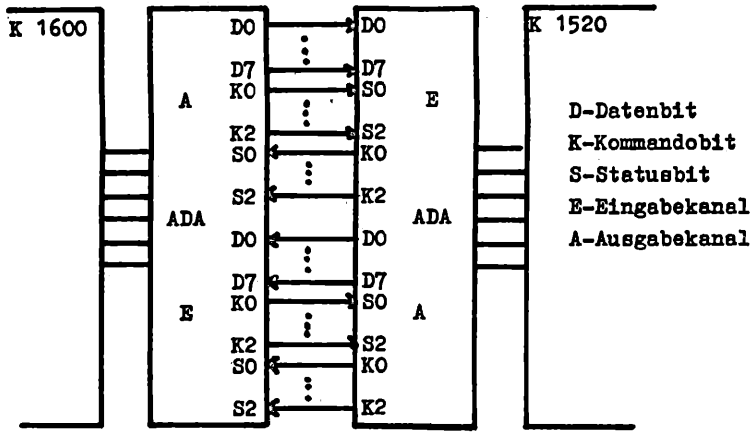


Abbildung 3: Kopplung der Anschlußsteuereinheiten der Rechner K 1600 und K 1520

Die Koppelsofware besteht aus einer Reihe von konkurierend arbeitender Prozesse. Diese Prozesse werden durch einen Kern gesteuert. Es gibt vier Systemprozesse (Transmitter, Receiver, Name-Server, Connection-Control) und mindestens einen Nutzerprozeß. Die genaue Stellung der Prozesse und ihrer Kommunikationspartner ist aus Abbildung 4 ersichtlich.
Die Aufgaben des Kerns bestehen in der Initialisierung des Gesamtsystems, der Organisation der parallelen Arbeit der einzelnen Prozesse und der Prozeßkommunikation. Dazu werden vom Kern folgende Prozeduren zur Verfügung gestellt:

CreatePort -Eröffnen eines Kommunikationspunktes
ClosePort -Schließen eines Kommunikationspunktes

CreatePacket	-Reservieren eines Speicherbereiches für ein Datenpaket
ClosePacket	-Rückgabe eines Speicherbereiches für ein Datenpaket an den Kern
SendMsg	-Senden eines Datenpaketes an einen Port (lokal)
WaitMsg	-Warten auf ein Datenpaket mit Zeitbegrenzung
CreateProcess	-Definieren eines Prozesses mit einer bestimmten Priorität
WaitInterrupt	-Warten auf eine Unterbrechung von einem Gerät
Init	-Starten des Gesamtsystems

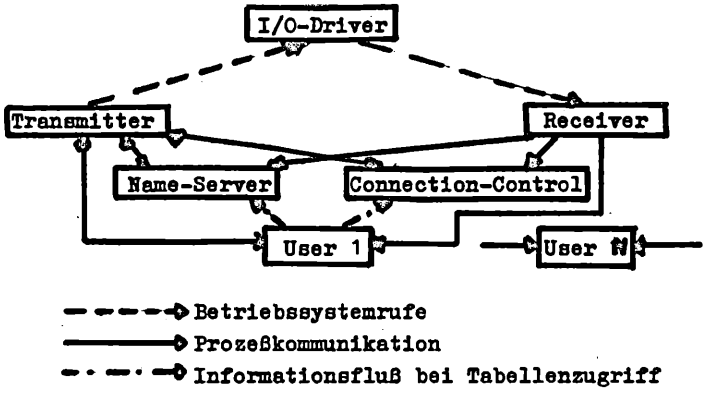


Abbildung 4: Prozeßstruktur der Koppelsoftware

Der I/O-Driver ist Bestandteil des jeweiligen Betriebsystems und wird mit den normalen Betriebssystemrufen durch den Transmitter bzw. Receiver aktiviert.
Der Receiver-Prozeß empfängt alle ankommenden Datenpakete und sendet sie dann mittels des Kern-Rufes SendMsg an den eigentlichen Empfängerport, der im Datenpaket spezifiziert ist. Der Receiver-Prozeß hat die höchste Priorität im Gesamtsystem und

ist interruptgesteurt (Kern-Ruf: WaitInterrupt).
Der Transmitter-Prozeß sendet Datenpakete an andere Rechner. Dazu übergibt er die modifizierten Datenpakete an den I/O-Driver. Wenn der I/O-Driver das Absenden des Datenpaketes bestätigt, wird das Datenpaket an den Absenderprozeß zurückgesendet. Der Transmitter-Prozeß hat die zweithöchste Priorität im Gesamtsystem.
Der Name-Server dient im Gesamtsystem zum lokalisieren der Kommunikationspunkte (Ports) von Diensten. Dazu wird in jedem Rechner eine Tabelle geführt, in der enthalten ist welche Dienste von den Nutzerprozessen des aktuellen Rechners den anderen Rechnern angeboten werden. Der Name-Server gibt eine Positiv-Meldung, wenn ein von einem anderen Rechner gesuchter Dienst in dem aktuellen Rechner zur Verfügung gestellt wird.
Der Connection-Control-Prozeß dient zur Überwachung von durch den Nutzerprozeß aufgebauten Verbindungen. Dazu werden vom Connectio-Control-Prozeß zeitzyklisch Testpakete an die jeweiligen Connection-Control-Prozesse gesendet, für deren Nutzerprozesse Verbindungen aufgebaut worden sind. Wird das Testpaket von dem entsprechenden Connection-Control-Prozeß in einer bestimmten Zeit bestätigt, so gilt die Verbindung als funktionsfähig, anderenfalls wird die Verbindung als abgebaut betrachtet und der Nutzerprozeß erhält beim nächsten Zugriff eine Fehlermeldung.
Ein Nutzerprozeß besteht aus zwei Komponenten, erstens den Systemroutinen und zweitens den eigentlichen Nutzerroutinen, die entweder einen Dienst des Systems anbieten oder einen Dienst des Systems nutzen. Abbildung 5 gibt eine Übersicht über die Struktur eines Nutzerprozesses. Die Systemroutinen des Nutzerprozesses enthalten die für die Kommunikation mit den Systemprozessen notwendigen Routinen.
Die User-Transmitter-Routine dient zum Absenden von Datenpaketen an andere Rechner. Durch die User-Receiver-Routine wird auf Datenpakete gewartet. Die User-Name-Server-Routine dient zum Installieren von Diensten und zum lokalisieren von Diensten auf anderen Rechnern.

Die User-Connection-Control-Routine dient zum Auf- und Abbauen von Verbindungen, zum Senden und Empfangen von Datenpaketen über aktive Verbindungen und zum Testen der Funktionsfähigkeit von Verbindungen.

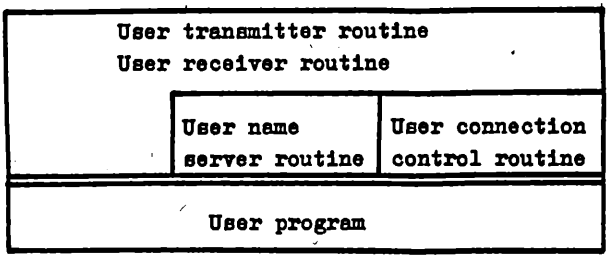


Abbildung 5: Struktur der Nutzersoftware

An eigentlichen Nutzerprogrammen wurden vorerst nur ein Filetransportdienst realisiert, der es erlaubt vom Bürocomputer gesteurte Dateien im FCS-Format auf dem K 1600 zu lesen und zu schreiben. Weitere Dienste sind möglich, so z.B. Dienste zum Drucken von Files für Bürocomputer, die keinen eigenen Drucker haben, Zugriff zu Dateien auf Minidisketten von Bürocomputern durch den K 1600.

Anschrift der Verfasser

Humboldt-Universität zu Berlin
Sektion Mathematik
Bereich Informationsverarbeitung
1086 Berlin
Unter den Linden 6
PF 1297

Bergholz, Gerhard

Zur Leistungsmodellierung von Rechnersystemen

0. Einführung

In diesem Beitrag werden die formelmäßigen Ergebnisse eines Zweischichtenbedienungsmodells für die Modellierung eines Einzelrechners vorgestellt, das auch als Grundlage für die Modellierung von Mehrrechnersystemen /9/ geeignet ist. In diesem Modell werden als Haupteinflüsse die Lastverteilungen über die geteilt genutzten und festzugeordneten Ressourcen erfaßt. Außerdem können mit diesem Modell solche Einflüsse wie Systemeigenlast (overhead) und ZVE:E/A-Überlappungen berücksichtigt werden. Die Berücksichtigung der Systemeigenlast wird im Vortrag behandelt.

Im Beitrag werden Durchsatz-, Reaktions- und Auslastungsverhalten als Einheit betrachtet, was die Analyse des Stapel- und Dialogbetriebs ermöglicht. Der letzte Teil des Vortrages beschäftigt sich kurz mit Ergebnissen der Leistungsmessung, Leistungsmodellierung und Leistungsverbesserung. Dabei wird eine kurze Übersicht über bereitgestellte Programmpakete gegeben.

1. Allgemeiner Modellansatz

Wir betrachten die typische Konfiguration eines Rechnersystems. Es werden von uns geteilt genutzte und festzugeordnete Ressourcen des Rechnersystems unterschieden. Festzugeordnete Ressourcen werden einem Job für die gesamte Abarbeitungszeit (Verweilzeit im Programmbereich) zugeteilt, während die geteilt genutzten Ressourcen verschiedenen Jobs zeitlich verschachtelt im Laufe der Jobabarbeitungszeit zugeteilt werden.

In unserem Modell betrachten wir als geteilt genutzte Ressourcen, die Einkanalressourcen Zentrale Verarbeitungseinheit (ZVE) und Selektorkanäle (SK1,...SK(n-1) und die Unendlichkanalressource (Infinite server) (Byte-)Multiplexkanal (MK). Die geteilt genutzten Ressourcen können entsprechend ihrer funktionellen Bedeutung auch als aktive Ressourcen bezeichnet werden.

Die festzugeordneten Ressourcen sind in unserem Modell alles Mehrkanalressourcen und ihrer funktionellen Bedeutung entsprechend passive Ressourcen. Als festzugeordnete Ressourcen werden in das Modell aufgenommen:

die Gesamtheit der Programmbereiche (PB) mit der Zahl m der Programmbereiche als Kanalzahl, der Hauptspeicher (HS) mit der Hauptspeicherkapazität m_s als Kanalzahl, die Gesamtheit der Magnetplattengeräte (MPG) mit der Zahl m_p der MPG als Kanalzahl, die Gesamtheit der Magnetbandgeräte (MBG) mit der Zahl m_b der MBG als Kanalzahl und die Gesamtheit der Drucker (PD) mit der Zahl m_d der PD als Kanalzahl.

Für die analytische Modellierung von Rechnersystemen mit dem Ziel einer Leistungsbewertung werden weltweit Bedienungsmodelle, meist Bedienungsnetze, genutzt. Ohne hier auf die Einzelheiten der Bedienungstheorie einzugehen, kann man für eine Rechnerkonfiguration bei Berücksichtigung der oben angeführten Ressourcen, die Grobstruktur des Rechnermodells als Zweischichtenmodell entsprechend Abb. 1 angeben. In diesem Zweischichtenmodell ist zu beachten, daß in der Modellschicht der geteilt genutzten Ressourcen hochfrequente Prozesse mit der Häufigkeit des E/A-Zugriffs und in der Schicht der festzugeordneten Ressourcen niederfrequente Prozesse mit der Häufigkeit der Jobaktivierung ablaufen. Die Wechselwirkung erfolgt mit der Frequenz der langsamen Prozesse der Schicht der festzugeordneten Ressourcen.

Neben der Rechnerkonfiguration besitzt im Bedienungsmodell der Rechnerinstallation der Arbeitslaststrom eine wesentliche Bedeutung. Ein Maß für den Umfang des Arbeitslaststroms ist der Verkehrswert ϱ des Gesamtsystems, für den gilt

$$\varrho = \lambda'/\mu' \tag{1}$$

mit der Ankunftsstromintensität λ' und der Bedienungsintensität μ' der Jobs. Hierbei gilt

$$\mu' = 1/\bar{t}_b' \tag{2}$$

mit der mittleren Summenbedienungszeit $\bar{t}_b'$ eines Job.

Außerdem sind die Lastverteilungen über die geteilt genutzten und über die festzugeordneten Ressourcen als Parameter des Arbeitslaststromes bedeutsam. Die Lastverteilung über die geteilt genutzten Ressourcen ergibt sich durch die Beschäftigungskoeffizienten x_i $(i=0,1,\ldots,n)$, wobei der Index 0 dem Prozessor, die Indizes $1,\ldots,n-1$ den entsprechenden Selektorkanälen und der Index n dem Multiplexkanal entsprechen. Dabei

gilt

$$g_i = g\, x_i \qquad (i = 0, 1, \dots, n) \tag{3}$$

$$\sum_{i=0}^{n} x_i = 1 \tag{4}$$

mit g_i als Verkehrswert der geteilt genutzten Ressource Nummer i .

Für die Lastverteilung auf die festzugeordneten Ressourcen werden die Bedarfskoeffizienten y_l , mit

$$y_l = \overline{N}_l / m_l \tag{5}$$

eingeführt. Dabei ist $\overline{N}_l$ der mittlere Kanalzahlbedarf eines Job der festzugeordneten Ressource l und m_l die verfügbare Kanalzahl der Ressource l, mit $l \in FR$, wobei FR die Menge der festzugeordneten Ressourcen

$$FR = \{PB,\ HS,\ HPG,\ HBG,\ PD\} \tag{6}$$

ist.

Als Bestandteile des Leistungsverhaltens einer Rechnerinstallation werden das Durchsatzverhalten, das Reaktionsverhalten und das Auslastungsverhalten betrachtet. Für das Durchsatzverhalten wird der Leistungsfaktor H, für das Reaktionsverhalten der Verweilzeitfaktor F und für das Auslastungsverhalten der Auslastungsgrad η des Gesamtsystems als Leistungsbewertungsgrößen betrachtet. Diese Leistungsbewertungsgrößen hängen von den Konfigurations- und Arbeitslastparametern ab. Der Zusammenhang zwischen den Konfigurations- und Arbeitslastparametern einerseits und den Leistungsbewertungsgrößen andererseits bildet das Leistungsbewertungsmodell der Rechnerinstallation.

2. Zweischichtenbedienungsmodell

Unter Nutzung der realen Eigenschaften der Rechnerinstallation, die aus Messungen gewonnen werden, kann, in Anlehnung an die näherungsweise vollständige Dekomposition (/1/,/2/), das Gesamtmodell eines Rechnersystems als ein Zweischichtenbedienungsmodell entwickelt werden (s. Abb. 1). Die Modellschicht der geteilt genutzten Ressourcen kann als erstes, unabhängig von der Modellschicht der festzugeordneten Ressourcen, berechnet werden. Anschließend ist dann die Berechnung der Modellschicht der festzugeordneten Ressourcen, unter Verwendung der Ergebnisse aus der Modell-

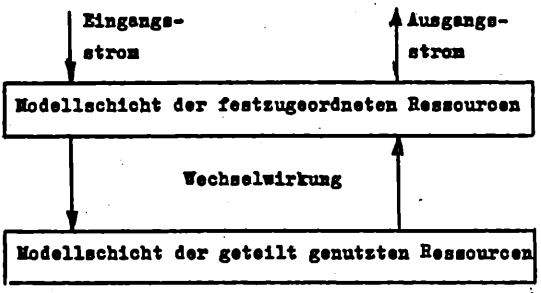


Abb. 1: Zweischichtenaufbau des Bedienungsmodells

schicht der geteilt genutzten Ressourcen möglich.

In der Abb. 2 wird eine allgemeine Übersicht über die funktionellen Zusammenhänge des Zweischichtenbedienungsmodells gegeben. Beide Modellschichten enthalten wiederum zwei Berechnungsstufen.

In der Modellschicht der geteilt genutzten Ressourcen stellt die erste Modellstufe eine Schar geschlossener Bedienungsnetze von Central-server-Typ dar. Aus den Beschäftigungskoeffizienten (Vektor $\underline{x}$) und der Zahl der geteilt genutzten E/A-Ressourcen n werden Zwischenwerte der Leistungs- und Verweilzeitfaktoren $R(k)$, $F(k)$ mit $k=1,2,\ldots,m$ berechnet.

Auf der Grundlage der Mittelwertanalyse (/3/,/4/) ergeben sich folgende Berechnungsformeln

$$F_i(k) = \begin{cases} 1 + x_i \dfrac{F_i(k-1)\cdot(k-1)}{F(k-1)} & \text{für ZVE, SK,} \\ 1 & \text{für MK,} \end{cases} \tag{7}$$

$$F(k) = \sum_{i=0}^{n} x_i F_i(k), \tag{8}$$

$$R(k) = k/F(k). \tag{9}$$

Hierbei sind

$$F_i(k) = \bar{T}_i(k)/\bar{T}_b \tag{10}$$

die Verweilzeitfaktoren der einzelnen geteilt genutzten Ressourcen.

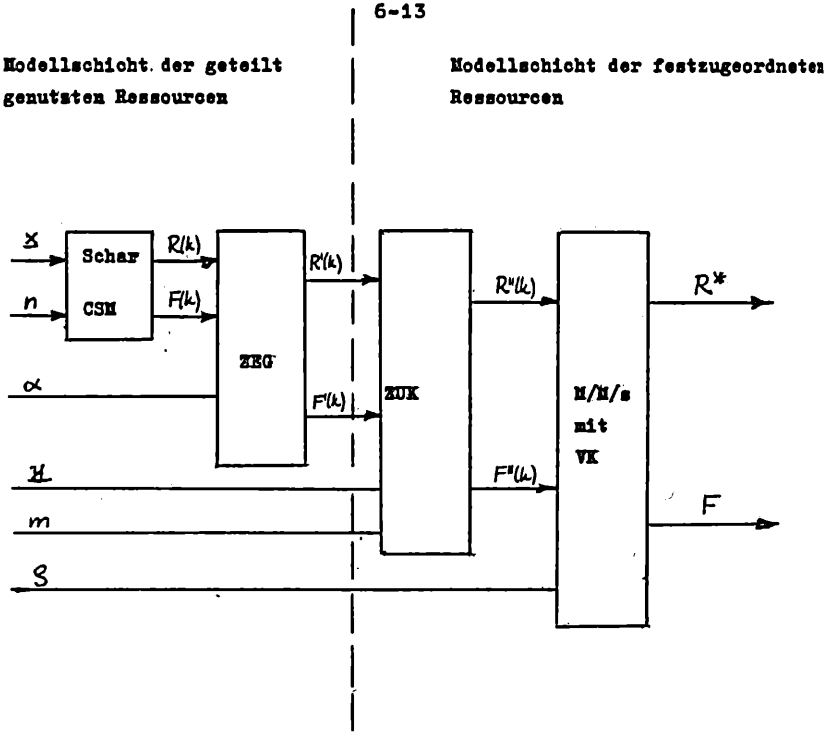


Abb. 2: Allgemeine Übersicht über die funktionellen Zusammenhänge des Zweischichtenbedienungsmodells

In der zweiten Berechnungsstufe (ZEG) der Modellschicht der geteilt genutzten Ressourcen können verschiedene Einflüsse, wie überlappte Arbeit innerhalb eines Job und Systemeigenlast (overhead) berücksichtigt werden. Wir beschränken uns hier auf den Einfluß der Systemeigenlast. Aus eigenen und fremden /5/ Messungen geht hervor, daß die Systemeigenlast von der Zahl der aktiven Jobs abhängig ist, wobei der lineare Ansatz

$$E(k) = 1 + \alpha \cdot (k-1) \tag{11}$$

für den von k abhängigen Anteil gilt. Dabei ist α der Anstieg der Systemeigenlast mit wachsender Jobzahl und $E(k)$ ist definitionsgemäß das Verhältnis von mittlerer Summenbedienungszeit bei einer bestimmten Aktivzahl k zur mittleren Summenbedienungszeit bei $k=1$.

In der Modellschicht der festzugeordneten Ressourcen berücksichtigt die erste Berechnungsstufe den Einfluß der Bedarfskoeffizienten y_l $(l \in FR)$ und der Zahl der maximal möglichen Programmbereiche m. Als Zwischenstufe wird eine blockierende Ersatzressource mit zufälliger Kanalzahl eingeführt, wobei die Verteilungsfunktion $\pi(s)$ dieser Kanalzahl von y_l und m abhängt (s. /6/). Auf dieser Grundlage kann als korrigierter Leistungsfaktor

$$R''(k) = \sum_{s=1}^{k-1} \pi(s) \cdot R'(s) + R'(k) \sum_{s=k}^{m} \pi(s) \tag{12}$$

aus dem im Modell der geteilt genutzten Ressourcen berechneten Leistungsfaktor $R(k)$ ermittelt werden.

In der letzten Berechnungsstufe, in der ein M/M/s-Bedienungssystem mit virtuellem Bedienungsknoten verwendet wird, kann der Verweilzeitfaktor F_m des Gesamtsystems nach folgenden Formeln rekursiv berechnet werden

$$\left.\begin{aligned} A_k &= [s/R''(k)] \cdot A_{k-1} \\ B_k &= A_k \cdot k + B_{k-1} \\ C_k &= A_k + C_{k-1} \end{aligned}\right\} \tag{13}$$

$$D_m = [1 + s/R(m)]^{-1} \tag{14}$$

und

$$F_m = \frac{B_{m-1} + A_m \cdot D_m \left[m + (s/R''(m)\, D_m\right]}{s\,(1 + C_{m-1} + A_m \cdot D_m)} \,. \tag{15}$$

Dabei gelten die Anfangswerte

$$A_0 = 1, \quad B_0 = 0, \quad C_0 = 0. \tag{16}$$

Die Formel (15) ermöglicht die Berechnung des Reaktionsverhaltens des Gesamtsystems. Das Durchsatzverhalten des Gesamtsystems ergibt sich mit

$$R^* = R''(m). \tag{17}$$

Für das Auslastungsverhalten des Gesamtsystems gilt

$$\eta = S/R^*. \tag{18}$$

Auf diesem Wege kann das Leistungsverhalten einer Rechnerinstallation berechnet werden.

3. Wesentliche Ergebnisse

In Ergänzung zu dem im Abschnitt 2 dargestellten Zweischichtenbedienungsmodell wurden an der TU Dresden in Zusammenarbeit mit dem DVZ Dresden Meßmittel geschaffen. Außerdem wurden die Modelle und Meßmittel als Grundlage für die Durchsatzerhöhung bei Rechnerinstallationen der Massendatenverarbeitung genutzt. Im Vordergrund stand dabei

- die optimale Verteilung der Systemdateien /7/
- die Jobnetzoptimierung /8/
- ein verbessertes SPOOL-Konzept /9/
  und
- die optimale Ablaufsteuerung /10/.

Auf den Ebenen der Leistungsmessung und -auswertung, der Leistungsmodellierung und der Anwendung für die Durchsatzerhöhung wurden an der TU Dresden eine Anzahl von Programmpaketen geschaffen. Eine Übersicht über ausgewählte Programmpakete ist in der Abb. 3 enthalten. Die gekennzeichneten Programmpakete wurden an Rechenzentren überführt. Dabei befinden sich Teile der Programmpakete (LEIST, RAMON, OPDA SISLAP) in allen 16 DVZ des KDV im produktiven Einsatz.

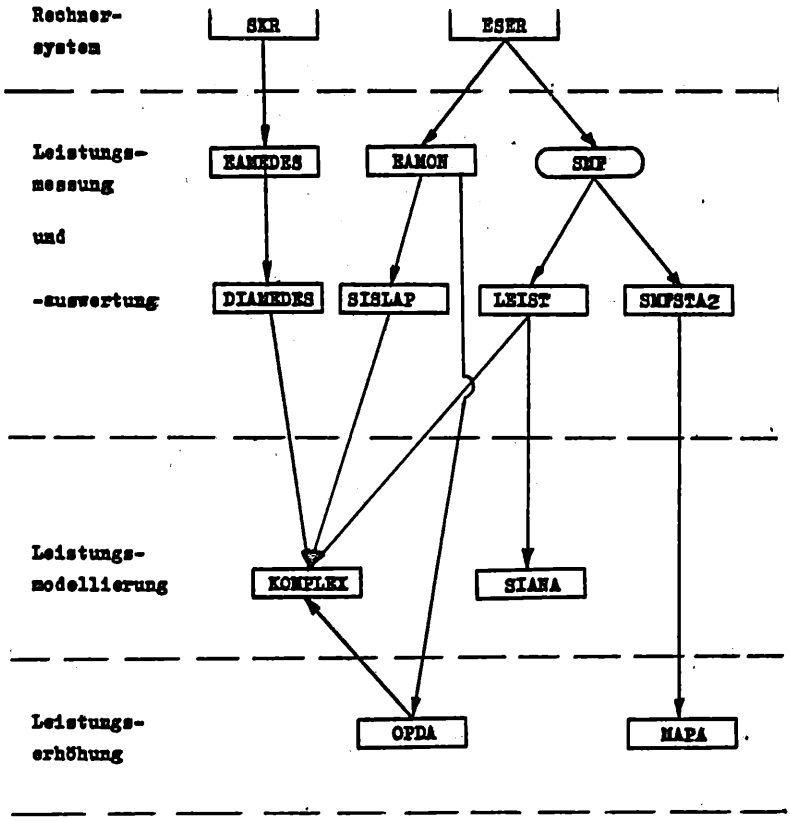


Abb. 3: Übersicht über ausgewählte Programmpakete der Leistungsmodellierung

Literatur

/1/ Curtois, P.: Decomposibility, Instability and Saturation in Multiprogramming Systems. Comm. ACM 18 (July 75) 371-377

/2/ Koroljuk, W.S.; Turbin, A.F.: Halbmarkowsche Prozesse und ihre Anwendung (russisch). Naukowa dumka Kiew (1976)

/3/ Reiser, M.; Sauer, C.H.: Queueing Network Models: Methods of solution and their Program Implementation. Current Trends in Programming Methodology vol. III: Software Modelling. Prentice-Hall Inc. Englewood Cliffs 1978. 115...167

/4/ Reiser, M.; Lavenberg, S.S.: Mean-value Analysis of Closed Multichain Queueing Networks. JACM 27 (1980) 2, 313-322

/5/ Awen, O.I; Gurin, N.N.; Kogan, Ja.A.: Gütebewertung und Optimierung von Rechnersystemen (russisch). Nauka Moskwa (1982)

/6/ Lehmann, D.: Analytische Verhaltensmodelle zur Leistungsbewertung im praktischen Rechenbetrieb. Dissertation A TU Dresden 1983

/7/ Bergholz, G.: Leistungsbewertung im Rechenbetrieb. rechentechnik/datenverarbeitung 21 (1984) H. 1

/8/ Bergholz, G.: Leistungsanalyse und Leistungserhöhung von Rechenanlagen. edv-spekte 2/82

/9/ Bergholz, G.: Zur Leistungsmodellierung von Mehrrechnersystemen (in Vorbereitung)

/10/ Mrosko, G.: Untersuchungen der Jobablaufsteuerung mit Hilfe hybrider Modelle. Dissertation A TU Dresden 1985

Verfasser:

Prof. Dr. sc. techn. Gerhard Bergholz

8019 Dresden, Heynahtsstraße 1a

Burghardt, Frank

SIMULATION VON KOMMUNIKATIONSPROTOKOLLEN

1. Einleitung

Mit der zunehmenden Verbindung von Computer- und Kommunikationstechnik in verteilten Rechnersystemen waechst die Notwendigkeit, sich waehrend des Entwurfs, Aufbaus und Betreibens solcher Systeme mit Fragen der Leistungs- und Zuverlaessigkeitsanalyse und des Systemdesigns auseinanderzusetzen. Dabei ist die Simulation auf der Grundlage diskreter Prozeßmodelle eine wichtige Methode, die Experimente mit sehr detaillierten Modellen gestattet.

Die Prinzipien der Nachrichtenuebertragung zwischen den Komponenten des verteilten Systems werden durch Kommunikationsprotokolle festgelegt. Diese sind Regeln ueber:
- die Formate und Arten der auszutauschenden Nachrichten,
- den zeitlichen Ablauf der Nachrichtenuebertragung,
- die Reaktion auf Fehler und den Wiederanlauf und
- den Mechanismus des Auf- und Abbaus einer Verbindung.

Analog den Prinzipien der strukturierten Programmierung hat es sich durchgesetzt, den komplexen Vorgang der Nachrichtenuebertragung in verschiedene Schichten mit funktioneller Arbeitsteilung zu gliedern, wobei jede Schicht zur Realisierung ihrer Dienste die der niederen Schichten in Anspruch nehmen kann. Dies fuehrt zu einem hierarchischen Schichtenmodell der Protokolle und erfordert natuerlich genau zu fixierende Schnittstellen zwischen den Schichten./1/

Protokolle koennen als Beschreibung der Wechselwirkung zweier Prozesse aufgefaßt werden, wobei deren Synchronisation durch die Kommunikation mit Hilfe der zu uebertragenden Nachrichten erfolgt. Diese paarweise auftretenden Prozesse seien Protokolleinheiten genannt. Die Protokolleinheiten reagieren beim Empfang einer Nachricht in Abhaengigkeit von ihrem Zustand (z.B. Zahl der unquittierten gesendeten Nachrichten, Zahl der freien Puffer) i.a. durch Senden einer Nachricht. Daher hat es sich bewaehrt, Modelle von Protokolleinheiten als Automatenmodelle zu realisieren.

Gute Unterstuetzung bei der Modellierung solcher Systeme bietet die verwendete Programmiersprache SIMULA, die erlaubt,
- Prozesse und ihre Kommunikation modellzeitlich nachzubilden
- und mehrere Objekte solcher Prozesse zu generieren.

2. Simulation von Protokollen

Fuer die Modellierung von Protokolleinheiten wurde ein allgemeiner Baustein bereitgestellt, der einige Standardprotokolloperationen sowie die Schnittstellen zu hoeheren und tieferen Schichten liefert. Zu den Operationen gehoeren:

- Empfang und Typanalyse von Nachrichten,
- Pruefen des Zustandes der Protokolleinheit,
- Fragmentieren und Blocken von Nachrichten,
- Verwalten der internen Zaehler der Protokolleinheit,
- Arbeit mit sogenannten time-outs zur Fehlerreaktion,
- Verwalten dynamischer Verbindungen (virtuelle Kanaele) und
- Generieren und Senden von Nachrichten.

Es gelang, mit diesem Apparat die konkreten Kommunikationsprotokolle HDLC, X.25 und AP 64 zu modellieren und entsprechend Experimente mit verschiedenen Protokollparametern durchzufuehren. Durch ein Tracesystem kann der Ablauf des Nachrichtenaustausches verfolgt werden.

3. Simulation von Kommunikationssystemen

Fuer die Leistungs- und Zuverlaessigkeitsanalyse eines Kommunikationssystems ist das Zusammenwirken von Hardware (Prozessoren, Uebertragungskanaele), Systemsoftware (einschließlich Kommunikationsprotokolle) und Arbeitslast (Nutzerauftraege) zu modellieren./2/ Hinzu kommt die Bereucksichtigung der Topologie eines solchen Systems. Deshalb wurden auch dafuer entsprechende Bausteine bereitgestellt./3/
Mit der Nutzung eines Dialogkonfigurators hat man die Moeglichkeit,
- beliebige topologische Strukturen von Kommunikationssystemen zu generieren,
- Parameterzuordnungen fuer die Hardware-, Software- und Arbeitslastmodelle vorzunehmen und
- das Zusammenwirken von Protokollen verschiedener Schichten zu testen.

Die Statistik des Simulationssystemes ist auf die Betrachtung von Durchsatz, Verweilzeiten und Auftreten von Uebertragungsfehlern ausgerichtet.
Es ist offensichtlich, daß der Rechenzeit- und Speicherplatzbedarf bei solchen detaillierten Simulationsmodellen erheblich ist. In /3/ werden auch Ansaetze zur Ueberwindung solcher Probleme beschrieben.

Literatur

/1/ Loeffler, H.
Rechnerverbundsysteme, Akademie-Verlag, Berlin, 1984
/2/ Wolfinger, B.
Modelle zur rechnergestuetzten Simulation von Kommunikationsfluessen in Rechnernetzen,
Diss., Techn. Hochschule Karlsruhe, 1979
/3/ Burghardt, F.
Ein Modellierungs- und Simulationskonzept fuer X.25-Paketvermittlungsnetze,
Diplomarbeit, Humboldt-Universitaet Berlin, 1983

F. Burghardt, Humboldt-Universitaet, Sektion Mathematik,
Bereich IV, 1086 Berlin, PSF 1297

Hammer, Rudolf
Dr. Kunicke, Manfred

Ein dynamisches Ressourcenüberwachungssystem für das Betriebssystem OS/ES

Für die Durchführung eines effektiven Rechenbetriebes ist es von großer Wichtigkeit, ständig die aktuellen Werte der Ressourcenbenutzung zur Verfügung zu haben. Insbesondere trifft dies für kritische Ressourcen, z. B. ZE-Zeit, Druckerpapier zu. Das im Betriebssystem OS/ES bereitgestellte System zur Erfassung der Ressourcenbenutzung (SMF) gestattet es zwar, sehr umfangreiche Analysen durchzuführen, diese sind allerdings nur über größere Zeiträume (eine Woche und mehr) sinnvoll.

Das von uns entwickelte dynamische Ressourcenüberwachungssystem wird parallel zum SMF angewandt und hat folgende Aufgaben zu erfüllen:

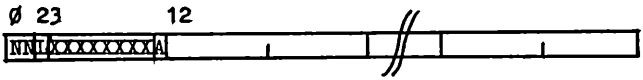
- Überprüfen der Legalität der Abrechnungsnummer
- Saldieren der Ressourcenbenutzung pro Job
- Mitteilen des Standes der Ressourcenbenutzung an den Nutzer nach Abarbeitung eines jeden Nutzerjobs
- Überwachung der geplanten Limite für die Ressourcenbenutzung.

Das dynamische Ressourcenüberwachungssystem ist so gestaltet, daß es Erweiterungen in Richtung eines abrechnungsnummernbezogenen Dateischutzes zuläßt. Es kann für die Ressourcenüberwachung sowohl im Stapelbetrieb als auch im Teilnehmerbetrieb eingesetzt werden.

Die programmtechnische Realisierung des Ressourcenüberwachungssystems basiert auf Ausgangsroutinen des Systemeingabeprogramms und des Programmlauforganisators sowie einer speziellen Separatorroutine. Für die Überwachung des Druckerpapierverbrauchs wurde eine Änderung der Routine IEFSDØ85 vorgenommen.

Alle für das Überwachungssystem notwendigen Daten (Abrechnungsnummern, Plan- und Istwerte der zu überwachenden Ressourcen, sowie in zukünftigen Ausbaustufen Angaben zum Datei-

schutz) stehen in der zentralen Nutzerdatei des Überwachungssystems. Diese befindet sich auf einem residenten Systemplattenstapel und ist eine Direktzugriffsdatei. Alle Daten, die zu einer Abrechnungsnummer gehören, sind zu einer Nutzereintragung zusammengefaßt und stellen ein Satz fester Länge der Nutzerdatei dar. In der Ausbaustufe ohne Dateischutz hat eine Nutzereintragung folgendes Format:



NN – relative Blocknummer der Nutzereintragung in der Nutzerdatei

L – L+1 = Anzahl der signifikanten Stellen der Abrechnungsnummer

XXXXXXXX – maximal achtstellige Abrechnungsnummer

A – Anzeiger, werden benutzt, um erreichte Limite bei kritischen Ressourcen zu signalisieren.

Die Länge einer Nutzereintragung ist abhängig von der Anzahl der im jeweiligen Rechenzentrum zu überwachenden Ressourcen. Für jede Ressource sind in einer Nutzereintragung zwei Worte vorgesehen. Es können bis zu acht Ressourcen überwacht werden.

Zur Gewährleistung eines schnellen Zugriffs zu den Nutzereintragungen ist eine Steuertabelle vorhanden. Sie hat ebensoviele Eintragungen wie die Nutzerdatei. Jede Eintragung der Steuertabelle besteht aus den ersten zwölf Bytes der entsprechenden Nutzereintragung. Die Steuertabelle ist Bestandteil der Ausgangsroutine IEFUJV des Systemeingabeprogramms. Anhand der Abrechnungsnummer aus der JOB-Anweisung wird über die relative Blocknummer die absolute Adresse der Nutzereintragung CCHHR ermittelt und im gemeinsamen Bereich der Ausgangsroutinen allen weiteren Routinen des Ressourcenüberwachungssystems zur Verfügung gestellt.

Die Ausgangsroutine IEFUJV überprüft die Abrechnungsnummer der JOB-Anweisung, aktualisiert das Anzeigerbyte der Steuertabelleneintragung und kontrolliert die Limite der zu überwachenden Ressourcen. Sie teilt der Jobverwaltung mit, daß

ein Job nicht abzuarbeiten ist, wenn

- die Abrechnungsnummer nicht vorhanden bzw. nicht registriert oder
- eine der überwachten kritischen Ressourcen erschöpft ist.

In den Spalten 73...80 der JOB-Anweisung wird eine Diagnosenachricht ausgegeben und der Job endet mit JCL-ERROR.

Die Ausgangsroutine IEFACTRT wird zur Jobendebehandlung aktiviert. Die vom abgearbeiteten Job verbrauchten Ressourcen werden durch die Unterprogramme RESS1 .... RESS8 berechnet und von der Ausgangsroutine saldiert. Im Falle der Überschreitung eines Limits für eine der kritischen Ressourcen wird das entsprechende Anzeigerbit gesetzt. Die Unterprogramme RESS1 bis RESS8 werden vom Anwender programmiert und müssen als Programmabschnitte vorliegen. Die Ausgangsroutine erstellt eine Information über den Stand der Ressourcenbenutzung (Plan- und Istwerte). Diese wird vom Separatorprogramm des Ressourcenüberwachungssystems im Kopf der Druckerausgabe dem Nutzer mitgeteilt.

Für die Ressource ZE-Zeit steht das Standardprogramm RESS1 zur Verfügung.

Eine der wichtigsten kritischen Ressourcen ist Druckerpapier. Der Druckerpapierverbrauch kann nur durch Routinen des Systemausgabeprogramms registriert werden. Zur Überwachung der Ressource Druckerpapier kann der modifizierte Modul IEFSDØ85 in Verbindung mit dem Standardunterprogramm RESS2 benutzt werden.

Zur Generierung, Wartung und zum Abziehen der Nutzerdatei stehen die Dienstprogramme USERINIT, USERUT und USERDMP zur Verfügung.

Das Dienstprogramm USRINIT legt die Nutzerdatei an, Parameter sind dabei die Anzahl der Nutzereintragungen und der zu überwachenden Ressourcen.

Das Dienstprogramm USERUT gestattet es, Abrechnungsnummern hinzuzufügen oder zu streichen, Plan- bzw. Istwerte für die überwachten Ressourcen zu ändern und die Nutzerdatei auszudrucken.

Um möglichem Datenverlust vorzubeugen, kann mit dem Dienstprogramm USERDMP die Nutzerdatei auf das Archivband der Res-

sourcenerfassung (SMF) abgezogen werden. Alle Informationen der Nutzerdatei werden als Nutzererfassungssätze aufgezeichnet und können zur Wiederherstellung der Nutzerdatei mit Hilfe des Programms USERIMP benutzt werden.

Heckert, Roland

Spezifikation und Implementierung eines Sitzungsdienstes für Rechnernetze in der DDR

Das mit der volkswirtschaftlichen Entwicklung einhergehende schnelle Anwachsen der Informationsmengen führte neben der breiten Anwendung rechentechnischer Mittel auch zu der Forderung nach Möglichkeiten für ihren entfernten Verbund. Solche Möglichkeiten werden mit dem Aufbau von Rechnernetzen geschaffen. Mit der Bereitstellung von Rechnernetzlösungen haben Anwender ein Mittel zur Realisierung ihrer spezifischen Kommunikationswünsche (Lastverbund, verteilte Datenbanken, Nutzung lokal nicht verfügbarer Gerätetechnik u. ä.) zur Verfügung.
Im Kombinat Datenverarbeitung wird seit einer Reihe von Jahren an der Entwicklung von Rechnernetzsoftware gearbeitet. Diese Arbeiten haben mit der Bereitstellung der Computer Communication Software (CCS /1/) einen Stand erreicht, der sowohl im Rahmen des Rechnerkommunikationsnetzes als auch in Kombinaten und Betrieben außerhalb des KDV Rechnernetzanwendungen in volkswirtschaftlichem Maßstab zuläßt.
Mit der Spezifikation und Implementierung eines Sitzungsdienstes für Rechnernetze in der DDR unternimmt der VEB LfA einen weiteren Schritt auf dem Wege zur Bereitstellung standardisierter Rechnernetzsoftware. Diese Entwicklung erfolgt in Zusammenarbeit mit anderen Einrichtungen auf der Basis des OSI-Referenzmodells für den Verbund offener Systeme /3/,/4/ und entsprechenden Standards der Gemeinsamen Spezialistensektion Rechnernetze des ESER und SKR.

Ein Großteil der Kommunikationsfunktionen, die in Rechnernetzen zur Verfügung stehen, haben anwendungsunabhängigen Charakter. Es ist deshalb wichtig, dem Anwender eine solche Schnittstelle zum Netz zur Verfügung zu stellen, die es ihm gestattet die für seine spezifische Anwendung benötigten Funktionen auszuwählen sowie die Art und Weise der konkreten Realisierung der Kommunikationsfunktionen für den Anwender transparent zu gestalten.
Ein weiterer wesentlicher Aspekt bei der Entwicklung von Rech-

netzsoftware ist die Möglichkeit der Überwindung von Inkompatibilitäten der im Netz zu integrierenden unterschiedlichen Hardware sowie der zwischen den lokalen Systemen laufenden Kommunikationsprotokolle. Dies wird erreicht durch eine verbindliche Festlegung der Kommunikationsprotokollabläufe und Schnittstellen im Netz mit Hilfe von Standards.
Als international akzeptiert kann gegenwärtig das Modell der Internationalen Standardisierungsorganisation (ISO) für den Verbund offener Systeme (OSI) gelten. Hiernach läßt sich jede Verbundsoftware sinnvoll in 7 Ebenen, vom physikalischen Niveau der Kommunikation (1) bis hinauf zur Anwenderschnittstelle (7) einteilen, wobei in jeder Ebene bestimmte Funktionen unter Nutzung der Funktionen der darunterliegenden Schicht realisiert werden, die dem Nutzer dieser Ebene an der oberen Schnittstelle als Dienste zur Verfügung gestellt werden. Die innerhalb einer Ebene ablaufende Kommunikation zwischen den einzelnen am Verbund beteiligten Systemen werden Ebenenprotokoll genannt und sind in Ebenenprotokollstandards spezifiziert.
Die Kommunikation zwischen benachbarten Ebenen in einem System erfolgt durch den Austausch von Schnittstellenanweisungen, sogenannten Dienstprimitiven. Der Nutzer einer Ebene kann dadurch zielgerichtet Kommunikationsfunktionen dieser Ebene auslösen.
Die Dienstprimitive selbst und ihre sinnvolle Aneinanderreihung zur Realisierung von Kommunikationsaufgaben sind in entsprechenden Dienstdefinitionen für jede Ebene beschrieben. Durch die Standardisierung von Protokollen und Schnittstellenaktionen in jeder Ebene werden folgende Vorteile erreicht:

- Verschiedene Implementierungen des Protokolls einer Ebene sind kompatibel.
- Die Implementierung der Software aufeinander aufbauender Ebenen ist problemlos in verschiedenen Einrichtungen möglich.
- Verschiedene Implementierungen einer Ebene sind gegeneinander austauschbar.

Besonders der zweite Aspekt ist für die Rechnernetzentwicklung in der DDR von Bedeutung.
So wird gegenwärtig auf der Grundlage vorhandener und neuer allgemein verfügbarer Datendienste ein Transportdienst (Ebene 4) auf einem Mikrorechnersystem (TSR/A) implementiert, der als front/end processor jedem am Verbund beteiligten Rechner

oder Terminal vorgelagert ist. Im Sinne des OSI-Ebenenmodells "darüber" ist der Sitzungsdienst angesiedelt auf den im folgenden näher eingegangen wird. Parallel zur Entwicklung des Sitzungsdienstes wird an der Entwicklung eines Fileservice gearbeitet, der die Funktionen des Sitzungsdienstes nutzt und auf dessen Grundlage dem Anwender eine breite Palette von Netzdiensten zur Verfügung gestellt wird.
Der Sitzungsdienst nutzt die Dienste der Transportschicht. Dies sind im wesentlichem:

- die Ende-zu-Ende-Steuerung zwischen den Systemen, deren Anwendungsprozesse miteinander zu kommunizieren wünschen,
- die transparente Übertragung der Datenströme über eine Transportverbindung in beiden Richtungen,
- sowie die Loslösung von Kenntnissen über das konkrete unterliegende Netz.

In der Sitzungsschicht sind alle Dienste enthalten, die für die Kommunikation zwischen Anwendungsprozessen in einem Rechnernetz erforderlich sind ohne anwendungsspezifische Funktionen zu enthalten. Die Funktionen des Sitzungsdienstes bilden damit die Grundlage für Entwicklungsarbeiten für die Schichten 6/7 unabhängig davon, welche Netzdienste implementiert werden sollen.
Der Sitzungsdienst stellt an seiner oberen Schnittstelle Dienste für den organisierten und synchronisierten Datenaustausch zwischen kooperierenden, einen Dialog führenden Nutzern bereit.
Es werden Mittel für:

a) den Aufbau einer Verbindung mit einem anderen SS-Nutzer, den Datenaustausch mit diesem Nutzer in einer abgestimmten Art und Weise und die geordnete Auflösung der Verbindung,

b) die Abstimmung der Nutzungsrechte bestimmter Dienste für den Austausch von Daten, die Synchronisation und Auflösung der Verbindung sowie die Festlegung des Datenaustausches im Duplex- oder Halbduplexmodus

c) das Einfügen von Synchronisationspunkten in einem Dialog und, nach Fehlern, die Wiederaufnahme des Dialogs von einem Synchronisationspunkt an

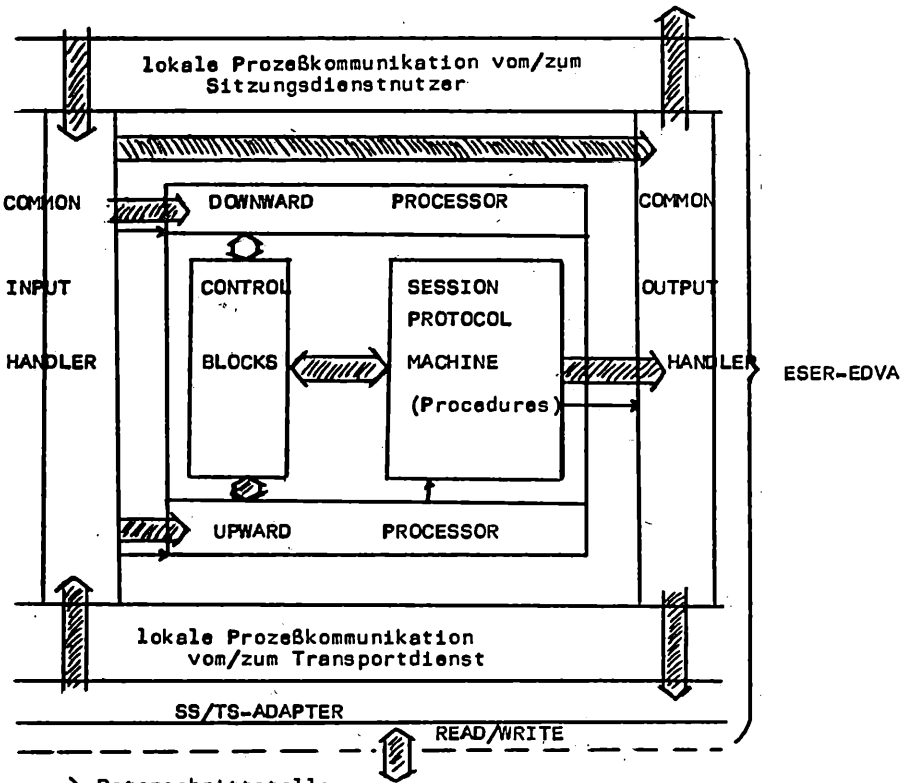
d) die Unterbrechung eines Dialogs und spätere Wiederaufnahme an einem vorher vereinbarten Punkt

bereitgestellt.
Bei Sitzungsverbindungsaufbau werden die Regeln des Dialogs zwischen den beiden kommunizierenden Nutzern des Sitzungsdienstes vereinbart. Dies geschieht durch:

- eine Abstimmung über die Auswahl von Teildiensten des Sitzungsdienstes für den Dialog auf dieser Verbindung (Auswahl und Vereinbarung von Funktionseinheiten)
- eine Abstimmung über das anfängliche Nutzungsrecht bestimmter ausgewählter Teildienste (Tokenkonzept) und
- die Vereinbarung einer bestimmten Dienstqualität.

Die Auswahl der Funktionseinheiten durch die Sitzungsdienstnutzer bestimmt die Möglichkeiten der Synchronisation ihrer Datenübertragung. In den Datenstrom können durch den Nutzer Synchronisationspunkte eingefügt werden, die eine Strukturierung des Datenstromes, eine ständige Kontrolle der Synchronität der kommunizierenden Nutzer sowie bei Fehlern Resynchronisation ermöglichen. In Abhängigkeit von den in einer Spezifikation oder Implementierung unterstützten Funktionseinheiten werden Subsets des Sitzungsdienstes unterschieden. Solche Subsets werden gebildet aus solchen Funktionseinheiten, die für einen bestimmten Anwendungstyp benötigt werden und unterscheiden sich im Grad ihrer Komplexität und Kompliziertheit wesentlich. In den Sitzungsdienststandards (/4/,/5/) werden drei solcher Subsets definiert.
Die seit 1984 im VEB LfA laufenden Spezifikationsarbeiten zum Sitzungsdienst und -protokoll erfolgen in Anlehnung an einen dieser Subsets, den Basic Synchronized Subset (BSS). Damit werden zum einen die wesentlichen, heute absehbaren Nutzerforderungen hinsichtlich der zu unterstützenden Teildienste abgedeckt, zum anderen wird damit die Voraussetzung geschaffen, Implementierungen mit diesem Leistungsumfang in den nächsten 2-3 Jahren, d. h. in überschaubaren Zeiträumen, zu realisieren. Ein erster Entwurf der Schnittstellenspezifikation im Umfang des BSS liegt vor. Die Protokollspezifikation wird mit Hilfe allgemeiner Beschreibungsmittel (endliche Automaten, PDL /6/, FDT /7/, PSDM /8/) vorgenommen, wobei eine entgültige Entschei-

dung für eine der Spezifikationstechniken noch nicht gefallen ist. Auf der Grundlage fertiggestellten Spezifikationsteile wird parallel dazu eine erste Implementierung des Sitzungsdienstes mit gegenüber der Spezifikation eingeschränktem Leistungsumfang auf ESER-EDVA vorgenommen. Diese Implementierung entspricht dem Basic Combined Subset für Halbduplexbetrieb und wird noch in diesem Jahr verfügbar sein. Einer späteren Erweiterung der unterstützten Teildienste steht bei sorgfältiger Umsetzung der Spezifikationsteile und entsprechenden logisch/funktionalem Konzept nichts im Wege. Die logisch funktionale Struktur der Sitzungsdienstsoftware ist in der folgenden Abbildung dargestellt.



Datenschnittstelle

Steuerungsübergabe

Abb.: logisch/funktionale Struktur der Sitzungsdienstsoftware und ihre Einbindung in die Implementierungsumgebung

Grob ist die Struktur in die Teile

- COMMON INPUT HANDLER
- SESSION PROTOCOL MACHINE und
- COMMON OUTPUT HANDLER

gegliedert.

COMMON deshalb, weil diese sowohl die Eingaben bzw. Ausgaben von/an den Transportdienst als auch die Sitzungsdienstnutzer behandeln.
Hereinkommende Anweisungen werden bei syntaktischer Richtigkeit in Abhängigkeit von ihrer Herkunft an den UPWARD (von TS)- oder DOWNWARD- (von SS-Nutzer)-PROCESSOR weitergeleitet.
Diese sind Bestandteil der SESSION-PROTOCOL MACHINE und haben die Aufgabe diese Eingaben in Bezug auf die in einer konkreten Implementierung vorhandenen Beschränkungen hin zu prüfen sowie die parallele Verwaltung einer Vielzahl von Verbindungen zu organisieren. Die parallele Verwaltung von Verbindungen wird über Steuerblöcke und Steuerblockzeigerlisten realisiert.
Im Kern der SESSION PROTOCOL MACHINE werden dann die eigentlichen Aktionen zur Realisierung des Sitzungsprotokolls entsprechend der ISO-Protokollspezifikation ausgeführt.
Ausgaben zur Realisierung des Sitzungsprotokolls oder Anweisungen zur Meldung lokaler Fehler (Syntax u.ä.) werden im COMMON OUTPUT HANDLER generiert und an die jeweiligen Prozesse ausgegeben.
Der ADAPTER realisiert zum Sitzungsdienst hin die Übergabe bzw. Übernahme von Schnittstellenanweisungen von/an den TS, und zum TS hin den Lese/Schreibzugriff zum Transportdienst auf dem front/end-processor. Dieses Strukturkonzept ist so angelegt, daß keine der genannten Funktionen an eine spezielle Implementierungsumgebung gebunden ist. Damit stehen der Implementierung dieses Konzeptes in verschiedenen Hardware-/Betriebssystemumgebungen keine erkennbaren Schwierigkeiten entgegen.
Neben ESER ist als nächstes die Implementierung auf K1630 vorgesehen. Für die Programmierung ist die Sprache C vorgesehen, um die Portierung der Software zu erleichtern.

Literatur

/1/ Computer Communication Software, Artikelserie, Rechentechnik/Datenverarbeitung Heft 1/1985

/2/ ISO IS 7498
"Bezugsmodell für offene Rechnernetze"

/3/ Löffler,H.:
"Rechnerverbundsysteme", Akademie-Verlag Berlin 1984

/4/ ISO DIS 8326
"Verbindungsorientierte Sitzungsdienstdefinition"

/5/ ISO DIS 8327
"Verbindungsorientierte Sitzungsprotokollspezifikation"

/6/ König.H.: Heiner,N.
"Definierender Bericht der Protokollbeschreibungssprache PDL"
TU Dresden, Sektion 8, Januar 1985

/7/ ISO/TC 97/SC 16/N 1825
"A Formal Description Technique based on an extended state transition model"
INWG-Note 356, Mai 1984

/8/ Heymer,V.:
"Protocol and Service description method"
IIR/1984

Dipl.-Phys.
Heckert, Roland
VEB Leitzentrum für Anwendungsforschung Berlin
Betrieb des Volkseigenen Kombinats Datenverarbeitung
1156 Berlin
Jacques-Duclos-Str. 50-52

Klaus Koehler

Formatunabhaengige Daten-Ein-und-Ausgabe in FORTRAN OS/ES

1. Einleitung

In der praktischen Nutzung von Rechenanlagen fuer wissenschaftlich-technische Zwecke sowie fuer die Aus- und Weiterbildung nimmt nach wie vor die Programmiersprache FORTRAN eine dominierende Rolle ein. Die Erfahrungen zeigen, dasz speziell die Behandlung der Daten- Ein- und -Ausgabe eine Reihe von Handhabungsmaengeln sowohl aus der Sicht der Anwendung als auch der Aus- und Weiterbildung zeigt. Die Ursachen liegen vor allem in dem starren FORMAT-Konzept begruendet :

- Hoher Ausbildungsaufwand
- Hoher Programmieraufwand
- Formatgebundene Dateneingabe entspricht nicht Dialogerfordernissen

Aus diesem Grunde wurden die nachfolgenden Konvertierungsroutinen mit folgenden Zielstellungen konzipiert :

- Formatfreiheit fuer Eingabedaten
- Standardformate fuer Ausgabedaten
- Unterstuetzung der E/A von CHARACTER-Daten
- Wegfall von FORMAT-Anweisungen im Anwenderprogramm
- Portabilitaet fuer Anlagen mit FORTRAN-Compilern

2. Konzeption der formatunabhaengigen E/A

2.1 Numerische Daten

Die Dateneingabe wird durch ein FUNCTION-Unterprogramm realisiert. Der Inhalt einer Datenkarte von 80 Zeichen Laenge wird zunaechst in einen Eingabebereich im Format (80A1) eingelesen und beginnend ab Spalte 1 auf Zeichen ungleich 'Blank' untersucht. Ab dieser 'aktuellen Position' wird versucht, die nachfolgende Zeichenkette als reelle Variable zu interpretieren.

Dabei werden eventuell vorhandene Sonderzeichen 'Dezimalpunkt', 'Plus' bzw. 'Minus' mit einbezogen. Die Sonderzeichen muessen entsprechend der FORTRAN-Syntax positioniert sein. Die Analyse und Konvertierung endet beim Auftreten von 'Blank' oder 'Komma' bzw. beim Erreichen der Spalte 80. Im Anwenderprogramm erfolgt die Zuweisung und gegebenfalls Typumwandlung des Funktionswertes an die entsprechende Variable.
Bei Wiedereintritt in die FUNCTION wird die Analyse an der aktuellen Position fortgesetzt. Wird bei der Konvertierung des Eingabebereichs die Spalte 80 erreicht, folgt das Einlesen einer neuen Karte und das Einstellen der aktuellen Position = 1.
Eine gesonderte Fehlerbehandlung erfolgt beim Erkennen unzulaessiger Zeichen im Eingabebereich, bei Zahlenbereichsueberschreitung, E/A-Fehlern und Erreichen des Dateiendes vor dem logischen Ende.

Die formatunbhaengige Ausgabe im Standardformat ist als SUBROUTINE konzipiert. Durch ihren Aufruf wird der Argumentwert vom Typ INTEGER*4 bzw. REAL*4 in ein Standardformat konvertiert und in einen Ausgabebereich von 120 Zeichen Laenge beginnend ab Druckposition 1 gestellt. Der Ausgabebereich wird gedruckt wenn die Postion 120 erreicht ist . Die Konvertierung von reellen Groessen erfolgt in Abhaengigkeit von ihrem Wert entweder in das F- oder E-Format.
Eine Fehlerbehandlung erfolgt bei E/A-Fehlern oder bei Erreichen des Dateiendes.

2.2 CHARACTER-Daten

Der Funktionsaufruf zur Eingabe von Zeichenkettendaten realisiert die Zuweisung des an der aktuellen Position befindlichen Zeichens (unabhaengig vom Wert) an eine Variable im Anwenderprogramm.
Bei der Ausgabe wird der Argumentwert als Zeichenkette im Format (A1) in die aktuelle Position des Ausgabebereichs gestellt. Eine Konvertierung erfolgt nicht.

2.3 Sonderfunktionen

Es ist moeglich, bei der Eingabe nach der Konvertierung sofort auf eine neue Karte ueberzugehen, bevor die Position 80 des Eingabebereichs erreicht ist.
Zur Gestaltung der Ausgabe werden folgende Sonderfunktionen bereitgestellt :

- Druck der Zeile vor vollstaendiger Belegung des Ausgabebereichs
- Ausgabe Leerzeile(n)
- Vorschub auf neue Seite
- Einfuegen von Leerzeichen in den Ausgabebereich ab aktueller Position

3. Realisierung

Die Konvertierungsroutinen fuer formatunabhaengige Daten- Ein- und -Ausgabe sind in FORTRAN geschrieben und damit prinzipiell fuer alle Anlagen verfuegbar, die einen FORTRAN-Compiler besitzen. Im Betriebssystem OS/ES erfolgt die Eingabe ueber die Datei 5 und die Ausgabe ueber die Datei 6.
Der Speicherbedarf liegt im OS/ES fuer die Eingaberoutinen bei ca. 3 K Bytes und fuer die Ausgaberoutinen bei ca. 4 K Bytes.

Folgende Routinen stehen z.Zt. zur Verfuegung :

GET	- Eingabe einer numerischen Variablen*
GETLN	- analog GET , Uebergang zu neuer Datenkarte
GETC	- Eingabe einer Zeichenkettenvariablen
GETCLN	- analog GETC, Uebergang zu neuer Datenkarte
PUTN	- Ausgabe einer REAL-Variablen (FORMAT (1X,F11.5))
PUTNLN	- analog PUTN , Druck Ausgabebereich
PUTI	- Ausgabe einer INTEGER-Variablen (FORMAT(4X,I8))
PUTILN	- analog PUTI , Druck Ausgabebereich
PUTC	- Ausgabe Zeichenkettenwert (FORMAT(A1))
PUTCLN	- analog PUTC , Druck Ausgabebereich
LESEN	- Einlesen naechste Datenkarte
LINE	- Druck Ausgabebereich bzw. Leerzeile

BLANK - Einfuegen von n Blanks in Ausgabebereich
PAGE - Vorschub auf neue Seite

4. Beispiel

```
C  DEMONSTRATIONSBEISPIEL FORMATFREIE E/A
      REAL A,B,C,DELTA,TEXT,X,XO,X1,Y
      DIMENSION TEXT(80)
C
      DO 100 I=1,80
        TEXT(I)=GETC(DUMMY)
        IF ( I .EQ. 40 ) CALL LESEN
100   CONTINUE
      A     =GET(DUMMY)
      B     =GET(DUMMY)
      C     =GET(DUMMY)
      XO    =GET(DUMMY)
      X1    =GET(DUMMY)
      DELTA=GET(DUMMY)
      DO 200 I=1,80
        CALL PUTC(TEXT(I))
        IF ( I .EQ. 40 ) CALL LINE
200   CONTINUE
      CALL LINE
      X=XO
      I=1
250   CONTINUE
      IF(.NOT.(X.LE.X1)) GOTO 300
        Y=A*X**2 + B*X + C
        CALL PUTI(I)
        CALL PUTN(X)
        CALL PUTNLN(Y)
        X=X+DELTA
        I=I+1
        GOTO 250
300   CONTINUE
      END
```

Datenaufbau
==========

```
FUNKTION Y= A*X**2 + B*X + C                    ( Karte 1 )
         I            X            Y            ( Karte 2 )
15,20,-3.2                                      ( Karte 3 )
-100     10000                                  ( Karte 4 )
  10                                            ( Karte 5 )
```

Druckbild
=========

```
FUNKTION Y= A*X**2 + B*X + C
         I            X            Y
           1   -100.00000   .14800E+06
           2    -90.00000   .11970E+06
           :            :            :
          11      0.0         -3.20000
          12     10.00000   1696.80000
           :            :            :
```

5. Zusammenfassung

Die beschriebenen Konvertierungsroutinen erlauben den Wegfall der formatgesteuerten Ein-/ Ausgabe im Anwenderprogramm. Die formatfreie Eingabe ist vor allem fuer Dialogprogramme sehr nutzerfreundlich. Die Portabilitaet der Routinen ist fuer Anlagen mit FORTRAN-Compiler problemlos gewaehrleistet.

Verfasser : Dipl.-Ing. Klaus Koehler
Ingenieurhochschule Zittau
Abt. EDV und Rechentechnik
8800 Zittau
Th.-Koerner-Alle 16

König, Hartmut
Heiner, Monika

P D L - Ein sprachlich-orientierter Ansatz zur Spezifikation und Verifikation von Rechnernetzprotokollen

1. Einleitung

Der Spezifikation von Rechnernetzprotokollen wird gegenwärtig große Aufmerksamkeit entgegengebracht. In der Literatur werden verschiedene Ansätze vorgeschlagen, die sich in einer Vielzahl von Methoden /Sunshine 82/,/Rudin 83/ niederschlagen. Trotz dieser Vielfalt hat bisher keine Methode eine breite Anwendung gefunden.

Dafür gibt es mehrere Gründe. Viele Methoden haben noch experimentellen Charakter und bedürfen weiterer Entwicklung. Auch solche Gründe wie das Fehlen von definierenden Berichten und Handbüchern, wie sie z.B. bei Programmiersprachen üblich sind, sowie der Unterstützung durch Organisationen müssen genannt werden. Ein sehr wesentlicher Aspekt ist auch die Nutzerakzeptanz. Viele Methoden sind unserer Meinung zu kompliziert, um eine breite Anwendung zu finden. Weitere Untersuchungen werden erforderlich sein, um den Anforderungen der Protokollspezifikation besser gerecht zu werden. Dabei ist es notwendig, mehr die Gemeinsamkeiten existierender Methoden zu nutzen als ihre Unterschiede herauszuarbeiten /Danthine 80/.

Die sprachlich-orientierte Spezifikation von Rechnernetz-Protokollen ist einer der wichtigsten Ansätze, die zur Zeit diskutiert werden /König 84/. Die Vorteile dieses Ansatzes, wie Exaktheit und Vollständigkeit der Beschreibung, Unterstützung einer schrittweisen Protokollentwicklung vom Entwurf bis zur Implementierung, sind i.a. anerkannt. Hauptansatz der Kritik an dieser Methode ist die Gefahr einer zu implementierungsnahen Beschreibung sowie die mögliche Belastung des Anwenders mit Details, die durch die Regeln der Sprache bedingt sind, aber keine Relevanz für die Protokollbeschreibung besitzen. Die her-

koemmlichen höheren Programmiersprachen sind jedoch nicht vorrangig für die Protokollbeschreibung entworfen worden und können somit auch nicht allen Anforderungen genügen. Wir sind der Meinung, daß hierfür Spezialsprachen entwickelt werden müssen, die die oben erwähnten Gefahren ausschließen.

Mit PDL (Protocol Description Language) /König 85a/ wird ein solches Sprachkonzept bereitgestellt, das auf modernen Erkenntnissen der Sprachentwicklung beruht. Es bildet den Kern eines Systems zur automatengestützten Protokollentwicklung /König 85b/, das den Entwurf, die Verifikation und die Implementierung von Rechnernetzprotokollen unterstützt. Im folgenden sollen ausgehend von den Erfordernissen der Protokollspezifikation sprachliche Aspekte der Gestaltung von PDL diskutiert werden.

2. Gestaltungsprinzipien von PDL

Die Gestaltung der Sprache ist durch die Anforderungen an die Protokollspezifikation bestimmt.

Die Hauptaufgabe der Protokollspezifikation ist die Dokumentation des Protokollentwurfs als Grundlage für eine einheitliche Interpretation des Protokolls bei der weiteren Realisierung. Der Protokollentwurf hat funktionellen Charakter und zielt nicht auf eine konkrete Implementierung. Zu seinen Aufgaben gehört z.B. die Bestimmung der Dienste einer Schicht, der möglichen Ereignisse und des Kommunikationsverlaufs. Zugleich wird mit der Spezifikation auch ein Mittel zur Verständigung zwischen mehreren an der Netzentwicklung beteiligten Partnern geschaffen. Ein weiterer Aspekt, der unserer Meinung in Anbetracht der zumeist verbal beschriebenen Protokoll-Standards sehr wichtig ist, besteht darin, daß eine (geeignete) formale Beschreibung wesentlich zur Anschaulichkeit und zum Verständnis der Protokollabläufe beiträgt.

Für die Gestaltung einer Beschreibungssprache ergibt sich daraus die Forderung, einfache und problembezogene Ausdrucksmittel zu schaffen, die das Wesen des Protokollablaufs widerspiegeln, ohne diesen mit implementierungsbedingten Details bzw. komplizierten Sprachregeln zu belasten. Die sprachlichen Mittel müssen

vertraut und leicht erlernbar sein und zugleich eine umfassende, eindeutige und gut lesbare Protokollbeschreibung gewährleisten.

Eine Protokollbeschreibungssprache soll nicht nur zur Dokumentation des fertigen Entwurfs dienen, sondern auch bei dessen Entwicklung Anwendung finden. Es muß möglich sein, gewisse Entwurfsentscheidungen zurückzustellen bzw. Entwurfsalternativen darzustellen.

Die Protokollspezifikation sollte auch die nachfolgenden Phasen der Protokollentwicklung, vor allem die Verifikation und die Implementierung, unterstützen. Das bedeutet, daß die Protokollbeschreibungssprache so zu gestalten ist, daß zum einen eine entwurfsbegleitende Verifikation und zum anderen eine relativ geradlinige Abbildung der Spezifikation auf die Implementierung möglich sind /Bolt 80/. Eine Automatenunterstützung dieser Prozesse ist wünschenswert und wird zunehmend angestrebt. Eine sprachlich geführte Protokollspezifikation bildet hierfür eine günstige Ausgangsbasis.

3. Sprachliche Aspekte der Gestaltung von PDL

Die Umsetzung obiger Gestaltungsprinzipien führte u.a. zu folgenden Entwurfsentscheidungen.

PDL versucht wichtige Begriffe des OSI-Referenzmodells /ISO 81/, wie z.B. Schicht, Einrichtung, Dienst und Protokoll, mit sprachlichen Mitteln zu fassen. Damit wird gleichzeitig ein sprachliches Modell für den OSI-Protokollmechanismus geschaffen.

PDL orientiert desweiteren auf eine Beschreibung des Protokollablaufs und nicht auf die Beschreibung von Zustandsübergängen, wie sie für viele Methoden üblich ist. Letztere (siehe z.B. /Bolt 80/) beschreibt i.a. das Verhalten von Einrichtungen mittels endlicher Zustandsautomaten. Eine solche Beschreibung widerspiegelt nicht immer eindeutig den Kommunikationsverlauf. (Es gibt hier gewisse Ähnlichkeiten mit der sogenannten goto-Programmierung.) Die PDL-Spezifikation dagegen erzwingt eine fortlaufende (goto-freie) Beschreibung der Wechselwirkung zwischen paaren Einrichtungen. Wir glauben, daß diese ablauf-

orientierte Darstellung natürlicher und adäquater den Protokollverlauf beschreibt.

PDL hat deskriptiven Charakter. Es enthält eine Reihe von Sprachelementen und -regelungen, die von vornherein eine implementierungsorientierte Darstellung verhindern. Die deskriptiven Elemente können entsprechend den aktuellen Beschreibungs- bzw. Abstraktionsniveau schrittweise verfeinert werden.

Im folgenden soll die Gestaltung der wichtigsten Elemente von PDL etwas ausführlicher betrachtet werden.

3.1. Die Schicht

Das OSI-Referenzmodell basiert auf einer Menge übereinanderliegender Schichten. Eine Schicht umfaßt einen logisch zusammenhängenden Bereich der Rechnernetzsoftware, der ein spezifisches Aufgabenspektrum erfüllt. Jede Schicht stellt der darüberliegenden einen Satz von Diensten zur Verfügung, die diese zur Realisierung ihrer Aufgaben nutzen kann. Die interne Funktionsweise einer Schicht ist autonom und nach außen nicht sichtbar. Damit kann eine Schicht prinzipiell durch eine andere ersetzt werden, falls die zugehörigen Anschlußbedingungen (Dienste) gewährleistet werden. Die Schicht (layer) wurde deshalb auch als Rahmen für die PDL-Spezifikation gewählt.

Vergleicht man die Charakteristika einer Schicht mit den Merkmalen abstrakter Datentypen, so stellt man eine weitgehende Übereinstimmung fest. PDL baut hierauf auf und realisiert die layer in Form eines abstrakten Datentyps. Die Dienste (service) der Schicht entsprechen dabei den Funktionen des abstrakten Datentyps, während ihre "Implementierung" durch die kommunizierenden Einrichtungen (entity) realisiert wird. Das Besondere dieses "abstrakten Datentyps" besteht darin, daß die "Implementierung" verteilt ist. Beispiel 1 zeigt den konkreten Aufbau einer layer.

Beispiel 1:

```
layer AB;
   service asked AB-SEND request(data: data block);
           responsed AB-RECEIVE indication;

   message PDU: record(data: data block; control bit: bit);

   protocol AB-Protocol: send<=>receive;
   entity AB-E;
      .   .   .
  #entity {AB-E}
#layer{AB}
```

3.2. Die Einrichtung

Die Einrichtungen bilden die "Kommunikationszentralen" der Schicht. Sie nehmen die Aufträge, die an die Schicht übergeben werden, an und veranlassen durch Wechselwirkung mit anderen Einrichtungen der Schicht ihre Erfüllung. PDL modelliert eine Einrichtung (entity) ebenfalls durch einen abstrakten Datentyp. (Diese Interpretation weist gewisse Ähnlichkeiten mit der Prozess-Auffassung in Concurrent Pascal /Brinch Hansen 77/ auf.) Die entity-Spezifikation besteht entsprechend aus einer Beschreibung der Komponenten der Einrichtung (Datenstrukturen, Timer), einschließlich der Schnittstelle (eintreffende Dienste und Nachrichten), und einem Aktionsteil der das dynamische Verhalten der Einrichtung festlegt. Für die Beschreibung des dynamischen Verhaltens der Einrichtung wurde mit der par event-Anweisung eine spezielle Variante der guarded regions /Brinch Hansen 79/ entwickelt. Die par event-Anweisung besteht aus einer Menge von Ereignis-Aktionsfolgen, die parallel ausgeführt werden können. Die Ausführung einer Aktionsfolge (i.a. die Ausführung eines Protokolls) wird durch Eintreten des zugeordneten Ereignisses (z.B. die Aktivierung eines Dienstes oder das Eintreffen einer Nachricht) ausgelöst. Solange kein solches Ereignis vorliegt bzw. nach Ausführung der Aktionsfolgen wartet die Einrichtung auf das Eintreten neuer Ereignisse.

Beispiel 2:

```
entity AB-E;
   service AB-SEND request;
   message PDU;
   protocol part send, receive;
   par event
      AB-SEND request: send |
      PDU <-- AB-E: receive
  #par
#entity {AB-E}
```

3.3. Dienste und Protokolle

Die Dienste bilden die Repräsentanten der Schicht nach außen. Ihre Spezifikation unterteilt sich in eine Nutzer- und eine Kontextspezifikation. Die Nutzerspezifikation, sie ist in PDL Bestandteil der layer-Spezifikation (siehe Beispiel 1), gibt an, wie ein Dienst zu aktivieren ist. In PDL geschieht das durch Angabe des Dienstprimitivs und der zugehörigen Parameter, wobei zwischen Diensten unterschieden wird, die von der oberen Schicht aktiviert werden (asked) und solchen, die in der eigenen Schicht initiiert werden (responsed). Die Kontextspezifikation beschreibt den Zusammenhang zwischen den Diensten. Sie ist z.Z. noch nicht in PDL enthalten. Hierfür sind unserer Meinung andere Darstellungsmittel erforderlich /König 85b/, die die sprachliche Spezifikation ergänzen müssen.

Ein Dienst wird innerhalb der Schicht durch die Funktionen der Schicht, die ihrerseits in den meisten Fällen Protokollabschnitte darstellen, und unter Inanspruchnahme der Dienste der darunterliegenden Schicht realisiert /ISO 81/. In PDL wird dieser Sachverhalt dadurch wiedergegeben, daß der Aufruf eines Dienstes, dargestellt als Ereignis in der par event-Anweisung, den Aufruf eines Protokoll-Teils(protocol part) bewirkt. Ein Protokoll-Teil (sie werden aus Gründen der Übersichtlichkeit im Anschluß an die layer-Spezifikation angeordnet) beschreibt den Kommunikationsverlauf aus der Sicht der Einrichtung (siehe Beispiel 3). Der Kommunikationspartner wird dabei durch eine

Beispiel 3:

```
protocol part send(entity receiver);
service AB-SEND indication;
message PDU to receiver,
        ACK from receiver;
timer t: 0..60;
begin
  .   .   .
  PDU --> receiver;
  start(t);
  wait event
    ACK <-- receiver: .   .   .
                      response AB-SEND indication;
                        .   .   .
    timeout t: skip
  wait;
  .   .   .
end {send}
```

fiktive Einrichtung, die im Kopf des Protokoll-Teils vereinbart wird, repräsentiert. Der reale Partner-Protokoll-Teil wird in der protocol-Spezifikation der layer festgelegt. Die Protokoll-Abarbeitung wird durch Aufruf eines Protokoll-Teils ausgelöst. Der zugehörige Partner-Protokoll-Teil wird durch Eintreffen einer Nachricht von der das Protokoll initiierenden Einrichtung angestoßen.

Obige Interpretation des Zusammenhanges zwischen Diensten und Protokollen, genaugenommen den Protokoll-Teilen, weist Ähnlichkeiten mit dem Prozedur-Aufruf auf und unterscheidet sich von der häufig anzutreffenden ausschließlichen Interpretation eines Dienstes als Nachricht. Im Unterschied zur Prozedur gibt es aber in dieser Interpretation keine Ergebnisparameter. Hierfür stehen, entsprechend dem OSI-Referenzmodell, die Dienste zur Verfügung, die durch die Schicht initiiert, die darüberliegende Schicht über bestimmte Ereignisse informieren (responsed-Dienste). Auch hier gilt die obige Interpretation. Der Aufruf eines solchen Dienstes (response) stößt in der da-

rüberliegenden Schicht einen Protokoll-Teil an bzw. bewirkt die Fortführung eines Protokolls.

3.4. Darstellung des Protokollablaufs

Ein Protokoll beschreibt die Wechselwirkung zwischen paaren Einrichtungen. In Wirklichkeit wird diese virtuelle Wechselwirkung durch Inanspruchnahme der Dienste der darunterliegenden Schicht realisiert. PDL trägt diesem Sachverhalt Rechnung und führt zwei Beschreibungsebenen - PDL/D und PDL/I - ein.

PDL/D (D=Design) konzentriert sich auf die Beschreibung der Kommunikation zwischen paaren Einrichtungen. Die Realisierung der Kommunikation durch die darunterliegende Schicht wird dabei nur beschränkt betrachtet. Die PDL/D-Spezifikation soll vor allem während der Entwurfsphase und zur anschaulichen Darstellung von Protokollabläufen Anwendung finden. Zu diesem Zweck wird PDL/D durch eine Reihe von Sprachregelungen unterstützt, auf die weiter unten noch eingegangen wird, die den deskriptiven Charakter dieser Spezifikation unterstützen.

PDL/I (I=Implementation) beschreibt die Kommunikation zwischen den paaren Einrichtungen unter Einbeziehung der Dienste der darunterliegenden Schicht. Es soll vor allem in der implementierungsvorbereitenden Phase eingesetzt werden. Die Spezifikation in PDL/I schränkt die Freiheiten, die die PDL/D-Spezifikation zuläßt, weitgehend ein und fordert in Hinblick auf die Implementierung konkrete Entscheidungen, ohne eine bestimmte Implementierung vorzuschreiben.

Der Protokollverlauf wird durch Protokollanweisungen beschrieben. Die wichtigsten Protokollanweisungen sind die Sende- (-->) und die <u>wait</u> <u>event</u>-Anweisung (siehe Beispiel 3).

Die <u>wait</u> <u>event</u>-Anweisung stellt wie die <u>par</u> <u>event</u>-Anweisung eine modifizierte Form der guarded regions dar. Sie beschreibt das Warten und Reagieren auf gewisse Ereignisse (Eintreffen einer Nachricht (<--), <u>timeout</u>). Im Gegensatz zur <u>par</u> <u>event</u>-Anweisung reagiert die <u>wait</u> <u>event</u>-Anweisung nur auf ein Ereignis.

Desweiteren gibt es Anweisungen für das Starten und Rücksetzen von Timern, für Wertzuweisungen, für das Eintragen (<:=) und Löschen (>:=) von Daten in komplexeren Datenstrukturen, z.B. Nachrichten, und andere. Für die Beschreibung des Protokollablaufs stehen die bekannten Steuerstrukturen höherer Programmiersprachen, wie die if-, case- und loop-Anweisung zur Verfügung. Außerdem besteht die Möglichkeit, gewisse Protokollaktionen nichtformalisiert mittels verbaler Konstruktionen zu beschreiben.

Die Spezifikation in PDL/D unterscheidet sich von der in PDL/I neben der bereits erwähnten Beschreibung der virtuellen Kommunikation (dargestellt durch --> und <--) durch eine Reihe von Reglungen für eine stärker deskriptive Darstellung. So gestattet PDL die implizite Einführung beliebiger elementarer Datentypen, die implizite Deklaration von Variablen in den Protokoll-Teilen und die wahlweise Angabe von Indizes in Datenstrukturen. Außerdem gibt es keine sprachlichen Mittel für die Bildung arithmetischer und logischer Ausdrücke. Hierfür müssen verbale Konstruktionen genutzt werden. Der deskriptive Charakter von PDL/D ist jedoch nicht auf die genannten Regelungen beschränkt. Für Aktionen mit einer festen Bedeutung wurde eine knappe und aussagekräftige Symbolik eingeführt. Genannt seien hier u.a. die Sende-Anweisung (-->), das Empfangsereignis (<--), der Aufruf eines responsed-Dienstes (response) und die Eintragungsanweisung (<:=) (Letztere fordert im Gegensatz zur Wertzuweisung, daß der Datentyp der rechten Seite lediglich einem der Komponententypen der komplexeren Datenstruktur auf der linken Seite entspricht.).

PDL/I ersetzt die virtuelle Kommunikation durch die Anpassung an die untere Schicht. Zu diesem Zweck werden die Sende-Anweisung und das Empfangsereignis auf die Dienste dieser Schicht zurückgeführt. In Vorbereitung einer Implementierung müssen die deskriptiven Elemente von PDL/D durch Einführung von Ausdrücken, Indizes, Datentyp-Definitionen u.a. verfeinert werden. Die Grundstruktur der PDL/D-Spezifikation bleibt jedoch erhalten.

3.5. Unterstützung der Verifikation

Die Gesamtkonzeption des in der Einleitung erwähnten Systems zur automatengestützten Protokollentwicklung sieht vor, die PDL-Spezifikation mittels Petri-Netzen zu verifizieren. Zu diesem Zweck wird die PDL-Spezifikation automatisch in ein Petri-Netz transformiert, das anschließend analysiert und simuliert wird (siehe hierzu /Heiner 84/). Auf diese Weise können Fehler im logischen Entwurf entdeckt werden. Der PDL-Sprachentwurf unterstützt diese Transformation durch eine Reihe von Festlegungen. Hier wären u.a. zu nennen

- die Schnittstellen-Spezifikation (service, message) in den entity's und den protocol parts, durch die die Synchronisationsstellen relativ einfach sondiert werden können,
- die angebotenen Steuerstrukturen, die klaren Petri-Netz-Strukturen entgegenkommen
- und nicht zuletzt die mittels der angebotenen bzw. erzwungenen Modularisierung mögliche Zerlegung bzw. Reduktion der Analysen.

4. Schlußbemerkungen

Die Diskussion der Gestaltungsprinzipien von PDL zeigt, daß der Protokollmechanismus des OSI-Referenzmodells unter Berücksichtigung moderner Erkenntnisse der Programmiersprachentwicklung und der parallelen Programmierung adäquat mit sprachlichen Mitteln beschrieben werden kann. Die Erfahrungen, die bei der Anwendung solcher Sprachen gesammelt werden, müssen zeigen, inwieweit die Kritik an der sprachlich geführten Spezifikation von Rechnernetzprotokollen noch gerechtfertigt ist.

Literatur

/Bolt 80/ Bolt Beranek and Newman Inc. (eds.): Formal Description Techniques for Network Protocols, INWG General Notes 257 (Dec. 1980)

/Brinch Hansen 77/ Brinch Hansen, P.: The Architecture of Concurrent Programs. Prentice Hall, New Jersey, 1977

/Brinch Hansen 79/ Brinch Hansen,P.: Distributed Processes - a new language concept. Com. of the ACM, Vol. 22 (1979), No. 4

/Danthine 80/ Danthine,A.A.S.: Protocol Representation with Finite-State Models, IEEE Trans. on Com., Vol. 28, No. 4, (1980)

/Heiner 84/ Heiner,M.;König,H.: Ein Ansatz zur Verifikation sprachlich spezifizierter Protokolle mittels Petri-Netzen, in Schriftenreihe WBZ MKR/IV, TU Dresden (im Druck)

/ISO 81/ ISO/TC 97/SC 16: Data processing - open systems interconnection - basic reference model. Draft Proposal ISO/DP 7498, 1981

/König 84/ König,H.: Ein sprachlich orientierter Ansatz zur Protokollspezifikation, in Schriftenreihe WBZ MKR/IV, TU Dresden (im Druck)

/König 85a/ König,H.;Heiner,M.: Definierender Bericht der Protokollbeschreibungssprache PDL. TU Dresden, Sektion Informationsverarbeitung, Forschungsbericht TU 08 RS-L/LN-K5/015 (1985)

/König 85b/ König,H.;Heiner,M.: The PDL-System - an unified Approach to the Specification, Verification, and Implementation of Protocols, eingereicht für COMNET '85, Budapest Oktober 1985

/Rudin 83/ Rudin,H.;West,C.H. (eds.): Protocol Specification, Testing, and Verification III, North Holland, Amsterdam, (1983)

/Sunshine 82/ Sunshine,C. (ed.): Protocol specification, Testing, and Verification, North Holland, Amsterdam, (1982)

Dr.-Ing. Hartmut König
TU Dresden
Sektion Informationsverarbeitung
8027 Dresden
Mommsenstr. 13

Dr.-Ing. Monika Heiner
IIR der AdW
1199 Berlin
Rudower Chaussee 5

Müller, K.; Piens, C.

Simulation von Verwaltungsstrategien für den virtuellen Direktzugriffsspeicher des Systems SPOOL 3

1. Ziele der Simulation

Mit der Einbindung eines virtuellen Direktzugriffsspeichers (s. /1/) in ein Betriebssystem - oder in ein mit diesem verbundenen Preschedulingsystem - kommt es zu einer Vergrößerung des Overheads. Dieser Overhead resultiert zum einen aus der ständigen Arbeit des Verwaltungssystems des virtuellen Direktzugriffsspeichers und zum anderen aus dem Aufwand zur Bereitstellung der benötigten Dateien im Primärspeicher. Durch die Simulation des virtuellen Direktzugriffsspeichers sollen im wesentlichen zwei Ziele verfolgt werden:

(1) Unterstützung einer sinnvollen Dimensionierung von Primär- und Sekundärspeicher,

(2) Erleichterung der Auswahl von Strategien bzw. Algorithmen für
- die Speicherplatzzuordnung,
- die Dateieinlagerung,
- die Dateiauslagerung,
- die Dateisicherung und
- die Reorganisation des Sekundärspeichers.

Gütekriterium soll der errechnete Gesamtaufwand für die notwendigen Dateitransporte zwischen den Speicherebenen sein, der einerseits in Spuren pro Zeitintervall und andererseits in Belastungszeit pro Zeitintervall gemessen wird.

2. Modellbeschreibung

Das Modell wurde in PASCAL programmiert. Es simuliert die Verwaltungskomponenten eines virtuellen Direktzugriffsspeichers nach /1/. Für jede Forderung, also für jedes Ansprechen einer Datei,sind der Standort der Datei zu bestimmen, die Benutzungsstatistik zu aktualisieren und falls erforderlich die nötigen Platzzuweisungs- und Transportoperationen durchzuführen. Durch Parametersteuerung können die Dimensionen des Primär- und des Sekundärspeichers, d.h. Anzahl und Kapazität der Magnetplatten und Magnetbänder, Simulationszeiträume und

Strategien für die Komponenten des Verwaltungssystems festgelegt werden.
Die Speicherplatzzuordnung erfolgt in zwei Schritten. Zuerst erfolgt die Datenträgerauswahl - sie kann zyklisch, nach dem größten Freibereich, oder nach der Gesamtanzahl von freien Spuren erfolgen -, danach die Zuordnung von Speicherplatz auf dem ausgewählten Datenträger. Diese kann nach den Methoden first-fit und best-fit erfolgen. Beim Scheitern der Methode kann die Zuordnung von max. fünf Bereichen (Extentbildung) ausgewählt werden. Zur Reduzierung der Speicherfragmentierung können die Dateigrößen normiert werden.
Für die Dateieinlagerung kann festgelegt werden, ob Anforderungen sofort erfüllt oder gepuffert werden, bis ein Zeitintervall abgelaufen ist oder eine bestimmte Anzahl von Anforderungen wartet.
Die Dateiauslagerung kann mit und ohne Bildung von Reservemengen erfolgen. Für die Festlegung der Reihenfolge der auszulagernden Dateien stehen mehrere Auswahlkriterien zur Verfügung:
- Tage seit dem letzten Zugriff,
- Zugriffshäufigkeit (bezogen auf einen bestimmten Zeitraum),
- Dateigröße (-größenklassen).

Durch Verknüpfung dieser Kriterien lassen sich noch weitere bilden.

Das Simulationsprogramm verarbeitet als Forderungsstrom speziell aufbereitete SMF-Sätze der Typen 14 und 15, die im realen Produktionsbetrieb am ORZ der Humboldt-Universität erfaßt wurden. Dabei wurden nur Zugriffe zu solchen Dateien erfaßt, die potentiell im virtuellen Direktzugriffsspeicher lokalisiert sein können.
Zur Auswertung der Simulationsexperimente wird eine Druckliste erzeugt, die für wichtige Modellparameter Summen und Mittelwerte enthält. Für die weitere Auswertung werden in eine Ergebnisdatei ereignisorientierte Meßsätze ausgegeben.

Im Modell werden erfaßt:
- Transportaufwand für Dateien einschließlich der Zeiten für die Magnetbandpositionierung,
- Arbeitslast des Systems,
- Speicherauslastung und -fragmentierung,
- Wartezeiten in Einlagerungswarteschlangen.

Im Simulationsmodell wird nicht berücksichtigt, daß das Verwaltungssystem des virtuellen Direktzugriffsspeichers selbst Ressourcen blockiert. Die Verwaltungsmaßnahmen und Dateitransporte haben auch keinen Einfluß auf den Forderungsstrom. Die berechneten Zeiten lassen qualitative Bewertungen des Verhaltens des virtuellen Direktzugriffsspeichers zu.

3. Arbeitslast

Es wurden die Dateizugriffe von 3556 Jobs innerhalb eines Zeitraumes von 54 Tagen erfaßt. In diesem Zeitraum wurde auf 434 potentiellen Dateien des virtuellen Direktzugriffsspeichers zugegriffen, die einen Speicherbedarf von 30 940 Spuren haben.
Eine tageweise Analyse ergibt folgende Werte:

	min	max	Mittelwert
Dateien	1	103	61,3
Spuren	100	10781	5789

Tabelle 1

Wichtig für die Abschätzung des zu erwartenden Aufwandes durch die Verwaltungsmaßnahmen des virtuellen Direktzugriffsspeichers sind die Größen der tageweisen Änderung der Dateizugriffe.

	min	max	Mittelwert
Dateien	1	87	40,6
Spuren	100	9551	3538

Tabelle 2

4. Simulationsergebnisse

In diesem Beitrag sollen lediglich Simulationsergebnisse bezüglich der Dimensionierung des virtuellen Direktzugriffsspeichers und der Auslagerungsstrategie dargestellt werden. Für alle Simulationsexperimente werden folgende Parameter fixiert:

- die Datenträgerauswahl erfolgt nach dem Kriterium des größten Freibereiches,
- die Speicherplatzzuordnung erfolgt nach der Methode best-fit mit Extentbildung ohne Normierung der Dateigröße,
- Einlagerungsanforderungen werden 3 Stunden gepuffert,
- die Dateisicherung erfolgt alle 3 Tage, spätestens 7 Tage nach der ersten Modifikation einer Datei,
- die Magnetplatten des Primärspeichers sind vom Typ 5061,
- der Sekundärspeicher umfaßt 33 Magnetbänder mit einer Länge von 720 m (Typ 5017).

Um Aussagen über eine günstige Größe des Primärspeichers zu erhalten, wurde die Anzahl der Magnetplatten variiert. Wichtige Kennziffern sind in Tabelle 3 dargestellt.

MP	Einlagerung		Auslagerung		Gesamtaufwand/Tag (h)
	Dateien/ Tag	Spuren/ Tag	Dateien/ Tag	Spuren/ Tag	
1	53,5	5480	53,5	5509	2:01
2	38,2	3671	37,7	3679	1:33
3	24,8	2075	25,1	2096	1:05
4	14,7	1069	15,4	1121	0:41
5	9,1	576	8,6	593	0:28

Tabelle 3

Wie zu erwarten, sinken die nötigen Aufwände mit steigender Größe des Primärspeichers. Interessant ist das Ergebnis eines Versuches, bei dem die Anzahl der Magnetbänder des Sekundärspeichers auf 70 erhöht wurde, deren Länge jedoch mit 350 m beträgt. Für 4 Magnetplatten sinkt der Gesamtaufwand auf 0:25 h.

Das zeigt, daß der Aufwand zum Suchen einer Datei im Sekundärspeicher, d.h. die Bandpositionierzeit, einen hohen Anteil am Gesamtaufwand hat. Bei 720 m langen Bändern und einer Bandgeschwindigkeit von 2 m/s beträgt er ca. 80 %.
Das Zusammenfassen von 3 Plattenstapeln zu einem entsprechend größerem brachte eine Senkung des Aufwandes um 2 % durch verringerte Speicherfragmentierung.
Für den Vergleich verschiedener Auslagerungsstrategien wurde ein Primärspeicher mit drei Plattenstapeln festgelegt.
Tabelle 4 zeigt Ergebnisse einer Versuchsreihe.

Strategie	Einlagerung Dateien/ Tag	Einlagerung Spuren/ Tag	Gesamtaufwand Tag (min)	max. Tagesaufwand (min) ohne Reorg.
1	24,6	2066	61	139
2	20,3	2434	55	130
3	22,1	2227	56	115
4	22,9	1617	56	128
5	17,4	2169	50	114
6	20,9	1870	53	150

Tabelle 4

Es wurden folgende Strategien getestet:

1 - Tage seit letztem Zugriff
2 - Tage seit letztem Zugriff * Größenklasse
3 - (Tage seit letztem Zugriff)^2 * Größenklasse
4 - 1/Zugriffshäufigkeit
5 - Größenklasse/Zugriffshäufigkeit
6 - Tage seit letztem Zugriff * Größenklasse/Zugriffshäufigkeit

Die Zugriffshäufigkeit wurde aus den Zugriffszahlen errechnet und zwar so, daß der jeweils letzte Zeitraum mit dem größten Gewicht eingeht.

Die Wahl der Auslagerungsstrategie hat einen deutlichen Einfluß auf den Gesamtaufwand, was in Differenzen von mehr als 20 % zum Ausdruck kommt. Die Anzahl der bewegten Dateien be-

einflußt den Aufwand stärker als ihr Umfang. In dieser, wie auch in anderen Versuchsreihen ist die Strategie die beste, die von der Zugriffshäufigkeit bezogen auf Dateigröße ausgeht.
Die Pufferung von Einlagerungsforderungen beeinflußt den Aufwand ebenfalls deutlich. Das Sammeln von Forderungen über jeweils sechs Stunden bringt gegenüber sofortiger Einlagerung einen um 20 % reduzierten Gesamtaufwand und 10 % weniger Dateitransporte.
In weiteren Versuchsreihen wurden unterschiedliche Kombinationen von Strategien getestet. Ferner wurde ein Forderungsstrom über einen längeren Zeitraum mit höherer Arbeitslast verwendet. Dabei zeigten sich in der Tendenz die gleichen Ergebnisse.

5. Schlußfolgerungen

Aus den hier dargestellten und weiteren Simulationsergebnissen lassen sich folgende Schlußfolgerungen ziehen:

- Der Aufwand für die Verwaltungsmaßnahmen des virtuellen Direktzugriffsspeichers hängt wesentlich von den Bandsuchzeiten bei der Einlagerung ab.
- Das Puffern von Einlagerungsanforderungen senkt deutlich die Bandsuchzeiten.
- Die Auslagerungsstrategie hat deutlichen Einfluß auf den Aufwand. Die Dateigröße muß bei der Bestimmung der Auslagerungsfolge berücksichtigt werden.
- Best-fit mit Extentbildung (eine im OS/ES übliche Art der Speicherzuordnung) ist die günstigste Methode der Speicherplatzzuordnung, wenn ausschließlich der Aufwand des Verwaltungssystems minimiert werden soll. Die Fragmentierung des Speichers muß näher untersucht werden.
- Sicherungsfristen unter 7 Tagen erhöhen deutlich den Aufwand, die technologische Sinnfälligkeit ist auch nicht offensichtlich.

Literaturverweise:

/1/ Kroß, G.: "Rationalisierung der Arbeit eines Rechnersystems durch die Integration eines virtuellen Direktzugriffsspeichers in das System SPOOL" (in diesem Heft)

Dipl.-Math. Conrad Piens
Dipl.-Math. Klemens Müller
Humboldt-Universität zu Berlin
Organisations- und Rechenzentrum
1086 Berlin, Unter den Linden 6

6-54

РЕПИН В.М.

СИСТЕМА КОЛЛЕКТИВНОГО ДОСТУПА ЭВМ МОСКОВСКОГО УНИВЕРСИТЕТА

В сообщении рассматривается система коллективного доступа ЭВМ для обеспечения научных исследований и учебного процесса. Система включает в себя дисплейную сеть и распределенные терминальные станции на базе мини-ЭВМ. В сообщении обсуждаются также вопросы развития системы коллективного доступа ЭВМ - применение стандартной модульной программируемой электроники и средне - частотных линий связи для передачи и обработки экспериментальных данных с измерительно-вычислительных комплексов.

Rieck, Klaus-Dieter

Datei- und Datenträgerverwaltung mit dem System EDAS zur ESER-Datei-Archivierung und -Sicherung

Das System EDAS dient speziell der Archivierung (Auslagerung) zeitweilig nicht benötigter Dateien, der Dateisicherung durch Magnetbandduplikate sowie der Aktivierung (Bereitstellung) wieder benötigter und der Rekonstruktion (Wiederherstellung) zerstörter Dateien. Dabei werden nicht, wie meistens üblich, periodisch alle (auch die vielen nichtgeänderten) Dateien gesichert, sondern nur die vom Datei-Eigentümer (Nutzer) als notwendig eingeschätzten.

Speicherhierarchie:

Die Dateien werden in einem mehrstufigen Speicherraum verwaltet.

Stufe 1: Magnetplatten, auf denen die Jobs die Dateien ansprechen.

Stufe 2: Magnetbänder zur Auslagerung der nichtaktiven Dateien; jeder Platte sind i. allg. mehrere Bänder zugeordnet.

Stufe 3: Magnetbänder für die Sicherheitskopien aller Plattendateien; sie werden im Zyklus chronologisch beschrieben, so daß jeweils ein Sicherheitsband aktuell ist.

Die Magnetbänder werden von EDAS selbst verwaltet (der Nutzer muß nicht die Archivnummern kennen); sind alle Dateien katalogisiert, kann auch die Zuweisung der Magnetplatten in Stufe 1 an EDAS übertragen werden.

Katalogführung:

Der EDAS-Katalog besteht aus dem Datenträger-, dem Nutzer- und dem Datei-Katalog; er enthält für jede Datei u. a. das Datum des letzten Zugriffs. Die Katalogdateien werden doppelt, auf zwei Systemplatten, geführt; dadurch wird eine automatische Rekonstruktion bei Beschädigungen oder Systemplattenwechseln möglich.

Kopiermodul:

Alle Kopiervorgänge werden von einem universellen Modul ausgeführt, der automatisch die geeigneten OS-Dienstprogramme aufruft, ohne daß der Nutzer Steuer- oder DD-Anweisungen angeben muß.

(Gegenwärtig verwaltet EDAS keine indexsequentiellen Dateien und keine Generationsgruppen.)

Kommandoprozessor:

Die EDAS-Dienste werden i. allg. durch eine einfache EXEC-Anweisung ohne DD-Karten aktiviert (Aufruf einer Prozedur, in der alle Kommandoprogramme angesprochen werden können). Die wichtigsten EDAS-Funktionen:

- Kopieren einzelner Dateien zwischen Stufe 1 und 2 bzw. 1 und 3 (COPY)
- Dateivorbereitung (PREPARE) mit Aktualisierung des Zugriffsdatums und ggf. mit Bereitstellung von Stufe 2 nach 1 (reicht der Speicherplatz auf der Magnetplatte nicht aus, wird automatisch durch Auslagerung der am längsten nicht benutzten Dateien Platz gewonnen)
- Archivieren (Auslagern nach Stufe 2) aller Dateien einer Magnetplatte, auf die nach einem bestimmten Datum nicht mehr zugegriffen wurde (ARCHIVE)
- Ausgabe von Verzeichnissen aller Dateien eines Datenträgers bzw. eines Nutzers (LIST) oder aller Datenträger (LISTVOL)
- Ändern von Namen und Zuordnungen (CHNAME,CHANGE)
- Verdichten von Magnetbändern (COMPRESS)
- Aufnahme von Datenträgern, Nutzern und Dateien in die EDAS-Verwaltung (PVOL,PUSER,INSERT)
- Entfernen aus der EDAS-Verwaltung (DVOL,DUSER,DELETE).
  Ein Teil der EDAS-Dienste kann auch über eine TSO-Kommandoprozedur im Dialog aufgerufen werden.

Ablaufsteuerung:

Auf Wunsch können die für den Nutzer wichtigsten EDAS-Funktionen auch mittels einfacher Zusatzinformationen im Job (Kommentarkarten) ausgelöst werden. Für diesen Fall ist der Start eines speziellen EDAS-Readers nötig.

Gegenwärtige Anwendungen von EDAS:

EDAS ist seit Anfang 1983 im Rechenzentrum der Hochschule für Verkehrswesen erfolgreich im Einsatz. Hauptanwendungsgebiet ist zur Zeit die Verwaltung von Testdateien in der Stapel- und Dialogarbeit.

Dabei konnte durch Auslagerungen (alle 1-2 Monate) eine virtuelle Vergrößerung der Magnetplattenkapazität um ca. 50 % erzielt werden. Außerdem wird EDAS zur Sicherung der Systemdateien benutzt; dadurch wird vermieden, daß vollständige Plattendumps hergestellt werden, obwohl nur wenige Dateien geändert wurden.

Begonnen wurde mit der Anwendung für das Projekt "Güterzugbildungsplanung", bei dem die einzelnen Teile des Streckennetzes durch eine Vielzahl von Dateien beschrieben werden, deren Verwaltung auf Magnetbändern manuell kaum zu bewältigen ist.
Weitere Anwendungsmöglichkeiten:

- ökonomische Speicherung der Dateien an ESER-Arbeitsrechnern, die an entfernten Rechnern eines Netzes beheimatet sind, aber zeitweilig am Host verfügbar sein müssen
- Sicherung von Dateien auf großen Magnetplatten bei Vermeidung vollständiger Dumps

Dr.-Ing. Klaus-Dieter Rieck, Hochschule für Verkehrswesen "Friedrich List", Rechenzentrum, 8010 Dresden, Friedrich-List-Platz 1

6-58

ЖЕЛЕЗНОВ Н.П.,РАЗУМОВА Г.С.,ФРОЛОВ А.К.

СРЕДСТВА СОПРОВОЖДЕНИЯ АРХИВА НА МАГНИТНЫХ НОСИТЕЛЯХ

Рассматриваются вопросы централизованного ведения архива на магнитных носителях и автоматизированного контроля за состоянием носителей.

Dr. Schreiber,Elke

Thema: "Optimale Leistungsplanung im Rechenzentrumsbetrieb"

Kurzvortrag:

Im Vortrag werden Mittel und Methoden der rechnergestützten optimalen Planung von Nutzeraufträgen der Massendatenverarbeitung vorgestellt.
Ausgangspunkt ist die Bestimmung von Plandaten für spezifische Arbeitslasten über ein implementiertes Meß-und Auswertesystem auf SMF-Basis.
Als geeignete planungstechnische Einheiten fungieren sogenannte Planungselemente,die eine Klassifizierung von Jobs nach spezifischen Kriterien darstellen.
Im Programmsystem OPAL ist ein mathematisches Modell zur optimalen Ressourcennutzung und termintreuen Abarbeitung der Planungselemente implementiert. Kernstück des Planungsalgorithmus ist ein Branch-and-Bound-Algorithmus,über den sich die Optimierung vollzieht.
Im Ergebnis werden optimale Jobmixe und dazugehörige Jobcharakteristika für die Ablaufsteuerung bereitgestellt. Steuervarianten moderner Betriebssysteme erfahren so eine notwendige Unterstützung.

STUCHLIK, FRANZ; WINZERLING, WERNER

AUTOMATISIERUNG DES BETRIEBES VON EDVA UND BEDIENERLOSER BETRIEB

1. EINLEITUNG

Die Situation in den Rechenzentren der DDR ist z. Z. dadurch gekennzeichnet, daß ihrer verfügbaren EDVA-Kapazität ein wesentlich größerer Bedarf an Rechnerleistung gegenübersteht. /33/ /40/
Das führte in der Vergangenheit dazu, daß in der DDR vorrangig die Erhöhung der extensiven und intensiven Auslastung der Rechenanlagen im Mittelpunkt wissenschaftlicher Arbeiten und Veröffentlichungen stand. /3/ /23/ /31/ /49/ /52/

Um die informationellen Prozesse bei der Durchsetzung der ökonomischen Strategie des X. Parteitages effektiv bewältigen zu können, wird die technische Basis der Rechenzentren, sowohl quantitativ, als auch qualitativ weiterentwickelt. /24/ /33/ /40/ /46/

Mit diesen Veränderungen wird eine umfassende und durchgehende Automatisierung aller Prozesse des Rechenzentrums notwendig, um auch bei größerer Rechnerkapazität und komplizierter werdender Ablauforganisation eine hohe Effektivität zu gewährleisten, ohne gleichzeitig den Personalbestand zu erhöhen.
Ein Übergang zu bedienerarmen Formen des Betriebes von Rechenanlagen wird dabei unumgänglich.

Während in der internationalen Literatur bereits über Forschungen zum Problemkreis "zeitweilig unbedient betriebene EDVA" berichtet wurde (/5/, /6/, /11/, /12/, /13/, /15/, /16/, /17/, /18/, /20/ /21/, /29/, /32/, /39/, /53/, /54/) findet diese Betriebsform in den Publikationen der DDR bisher kaum Beachtung, /23/ bzw. wird sie als ungeeignete Zielstellung für die Rechenzentrums-Automatisierung in der DDR abgelehnt (/31, S. 90/).

2. AUTOMATISIERUNGSTENDENZEN BEI MITTLEREN UND GROSSEN EDVA

Bei den Veränderungen auf dem Gebiet der Informatik wird allgemein davon ausgegangen, daß im Perspektivplanzeitraum Rechenzentren mit mittleren und großen EDVA ihre Bedeutung behalten, wenn auch mit einem veränderten Aufgabenspektrum. /10/ /36/ /40/ /44/ /45/ /51/

Im folgenden werden einige Entwicklungen und ihre Auswirkungen auf die Automatisierung des Rechenbetriebes diskutiert. /10/ /15/ /17/ /24/ /26/ /28/ /31/ /32/ /34/ /36/ /39/ /40/ /41/ /42/ /43/ /44/ /45/ /46/ /48/ /49/ /51/

(1) Steigende Leistungsfähigkeit der EDVA

Die steigende Leistungsfähigkeit der EDVA (höhere Operationsgeschwindigkeiten, größere und schnellere Hauptspeicher, schnellere Systemspeicher u. a.) /24/ /43/ vergrößert den möglichen Jobdurchsatz, und erhöht seine Komplexität.
Damit wird nur ein automatisierter Betrieb der Rechenzentren eine intensive Auslastung der Rechnerressourcen gewährleisten.

(2) Direkte Jobeingabe in die EDVA durch Stapelfernverarbeitung

Mit der Vergrößerung der Anzahl der Stapelfernverarbeitungs- und Dialogterminals, und entsprechender Steuerprogramme erlangt die Dialogstapelfernverarbeitung, z. B. in wissenschaftlichen und Hochschul-Rechenzentren der DDR, eine zunehmende Bedeutung /28/ /51/.
Zur Kontrolle und Einplanung der ferneingegebenen Jobs für die Stapelverarbeitung müssen automatisierte Lösungen angewendet werden, da diese Jobs nicht mehr als Lochkartenstapel die Arbeitsvorbereitung durchlaufen.

(3) Rechnernetze

Rechnernetze ermöglichen z. B. eine zunehmende Spezialisierung der angeschlossenen Rechenzentren /24/ /26/ /48/.
Auch hier durchlaufen die ferneingegebenen Aufträge nicht mehr die Arbeitsvorbereitung des Zielrechners. Die Kontrolle der Nutzungs- und Zugriffsberechtigung und die Einplanung zur Verarbeitung bedürfen automatisierter Lösungen.

(4) Speicherung und Verwaltung von Dateien

Die Entwicklung bei Direktzugriffsspeichern für mittlere und große EDVA führt zu Festplattenspeicher-Geräten mit Kapazitäten von mehreren GByte je Plattenstapel /1/ /5/.
Es wird davon ausgegangen, daß künftig leistungsfähige EDVA über ein Speichersystem mit 5 - 6 Hierachiestufen verfügen; vom schnellen Cache-Pufferspeicher, über den Hauptspeicher, den Plattenspeichern, einem Massenspeicher, bis zur letzten Hierachiestufe, dem Magnetbandarchiv. /42/
Für die Verwaltung der verschiedenen Speichermedien werden zunehmend - besonders in wissenschaftlichen und Hochschul-Rechenzentren - automatisierte Dateiverwaltungssysteme eingesetzt /19/ /31/. Dabei erfordert nur die letzte Hierachiestufe innerhalb des Speichersystems - das Magnetbandarchiv - künftig noch eine Bedienung durch Operator /34/ /39/.

Das erste international verbreitete automatisierte Dateiverwaltungssystem war der, im Betriebssystem GEORGE 3 integrierte, Datei-Speicher (GEORGE 3 Filestore) für ICL-Rechner. /4/
In der DDR ist die Entwicklung eines Dateiverwaltungssystems z. B. innerhalb des Systems SPOOL ab Version 3, als Konzept eines virtuellen Direktzugriffsspeichers vorgesehen. /34/

(5) Benutzergesteuerte Datenein- und -ausgabe

Die erwähnten Entwicklungen auf dem Gebiet der Nachrichtenübertragung, sowie der Mikrorechentechnik (z. B. lokale Netze) ermöglichen den Nutzern den dezentralen Zugriff zu zentralen Rechnerressourcen /24/ /40/ /48/ /51/.
Dabei werden zunehmend die Ein- / Ausgabeprozesse zur Jobeingabe und Ergebnisausgabe nicht mehr im Rechenzentrum ausgeführt, sondern direkt am Arbeitsplatz des Nutzers (oder in dessen unmittelbarer Nähe) und unter seiner Steuerung.

Wird Spezialperipherie (z. B. Plotter u. ä) auch dezentral aufgestellt und durch die Nutzer bedient, können in der Praxis auftretende Sonderfälle der Ein- / Ausgabe vom Nutzer behandelt werden. Damit ist es möglich, schwer zu automatisierende bzw. selten auftretende, Bedienhandlungen an speziellen peripheren Geräten, aus dem Rechenzentrum herauszunehmen und an die Nutzer zu übergeben.

Die geringfügige Erhöhung des Arbeitsaufwandes beim Nutzer, kann i. d. R. jedoch vernachlässigt werden, da
- die Datenerfassung in einem wissenschaftlichen bzw. Hochschul-Rechenzentrum meistens vom Nutzer selbst durchgeführt wird, und
- sich hierbei für den Nutzer die Durchlaufzeit seiner Aufträge durch das Rechenzentrum verkürzt.

(6) Zuverlässigkeit künftiger EDVA und Unterstützung der Rechenzentrums-Organisation

Mit dem Einsatz hochintegrierter Bauelemente und der Weiterentwicklung der Architektur von Rechenanlagen und ihrer Ressourcenverwaltung steigt auch deren Zuverlässigkeit /1/ /24/ /43/.
Künftige Betriebssysteme werden in zunehmendem Maße die gesamte Rechenzentrums-Organisation unterstützen /15/ /31/ /41/ /44/ /49/.

3. UNBEDIENTER BETRIEB VON EDVA IN RECHENZENTREN

Der Betrieb von EDVA für Stapelverarbeitung erfordert eine manuelle Bedienung.
Es gibt jedoch Perioden, in denen wenige oder keine Bedienhandlungen erforderlich sind. Durch technologische Maßnahmen ist es möglich längere Zeitabschnitte "ohne Bedieneingriffe" zu schaffen.
Aus der Literatur sind eine Reihe von Rechenzentren bekannt, die nach diesem Prinzip Stapel-EDVA die Nachtschicht bzw. das gesamte Wochenende ohne Anwesenheit von Operatoren betreiben. /5/ /6/ /9/ /12/ /16/ /17/ /18/ /20/ /21/ /23/ /29/ /32/ /39/ /53/ /55/

3.1. Unbedienter Betrieb von Stapel-EDVA in der DDR

Eine erste Variante eines bedienerarmen Betriebes von EDVA in der DDR publizierte 1982 das Rechenzentrum der Humboldt-Universität zu Berlin /23/.
Als Gründe für das bisher nur geringe Interesse an einer unbedienten Betriebsweise von Stapel-EDVA (/31, S. 90/) werden u. a. gesehen:

(1) Im Unterschied zu anderen Ländern wurde in der DDR der "Bediener einer EDVA" schon frühzeitig ein Ausbildungsberuf. Dadurch, und durch staatliche Lenkungsmaßnahmen standen bisher immmer genügend ausgebildete Operator zur Verfügung, während in anderen Ländern der Mangel an ausgebildeten Operatoren zum bedienerarmen bzw. unbedienten Betrieb der EDVA zwang. (vgl. /27, S. 2/ mit /16/, /22/, /29/)

(2) Aufgrund der Hardware-Kostenstruktur, sowie der Produktionsfondsabgaben ist der Anteil der Lohnkosten im Verhältnis zu den Hardwarekosten in der DDR geringer als in den Ländern, aus denen bedienerlos betriebene Stapel-EDVA bekannt geworden sind /12/ /16/.

(3) Eine Störung während des unbedienten Betriebes - wie z. B. ein Betriebssystemabsturz - kann zu erheblichen Produktionsausfällen führen, weil i. d. R. bisher kein automatischer Wiederanlauf einer unbedient betriebenen EDVA möglich ist /5/ /11/.
Eine hohe Auslastung der Anlagen kann dadurch gefährdet werden.

(4) Während des unbedienten Betriebes muß die Kommunikation zwischen "Operator" und Betriebssystem automatisiert werden. Dafür steht das System SPOOL 2.0 seit 1982 zur Verfügung /23/.

(5) Ein unbedienter Betrieb einer EDVA erfordert eine hohe Systemverfügbarkeit und Zuverlässigkeit, um die Ausfallzeiten gering zu halten (siehe auch Punkt 3).
In /14/, /31/ und /47/ werden Untersuchungen zur Störsicherheit von EDVA des ESER durchgeführt. Dort wird ausgewiesen, daß aufgrund von Störungen in der Gerätetechnik, sowie aufgrund von Mängeln in der Technologie des Rechenzentrums durchschnittlich 5,2 - 18 IPL's innerhalb von 24 Stunden Nutzzeit notwendig waren.

(6) Die Kapazität an Direktzugriffsspeichern - wie sie in den vielen Rechenzentren der DDR i. d. R. anzutreffen ist (/1/ /5/ /8/ /21/ /30/) - führt zu einer erhöhten Anzahl manueller Montierhandlungen. Dadurch werden die Abschnitte begrenzt, in denen ein unbedienter Betrieb der EDVA - ohne Montierhandlungen an Direktzugriffsspeichergeräten - möglich ist /23/.
Große Speicherkapazitäten auf Direktzugriffsspeichern sind eine notwendige Voraussetzung für eine Erhöhung der Effektivität und die Automatisierung des Betriebes von EDVA. /11/ /21/ /31/ /39/ /49/ /51/

3.2. Klassifizierung des unbedienten Betriebes

3.2.1. Klassifizierung nach der notwendigen Bedienung

Im folgenden wird der Betrieb einer EDVA nach dem Umfang der notwendigen Bedienung klassifiziert, ohne auf die "derzeitige Praxis" (im Bild 1 als "normaler bedienter Betrieb" bezeichnet) /31/ /38/ /50/ näher einzugehen.

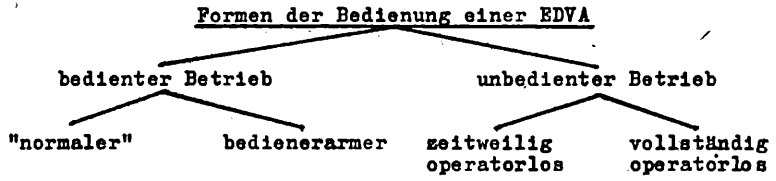


Bild 1: Klassifizierung nach dem Umfang der Bedienung einer EDVA

Bedienerarmer Betrieb einer EDVA

Durch Automatisierungsmaßnahmen können während des Betriebes einer EDVA Zeitabschnitte geschaffen werden, in denen keine Bedienereingriffe notwendig sind.
In praktischen Lösungen arbeitet hier die Zentraleinheit ausschließlich mit vormontierten magnetischen Datenträgern. Der Jobstrom und die Verarbeitungsergebnisse werden auf einem magnetischen Datenträger zwischengespeichert - "gespoolt" /12/ /23/. Bedienereingriffe sind nur nach einem Betriebssystem-Absturz notwendig, um den Neustart des Betriebssystems vorzunehmen.
Bei einer hinreichend hohen Systemverfügbarkeit der EDVA kann die Arbeit der EDVA auch ohne Anwesenheit eines Operators erfolgen.
Der zeitweilige Betrieb einer EDVA ohne Anwesenheit eines Operators, (jedoch auch ohne die Möglichkeit eines automatischen Wiederstarts der EDVA nach einem Betriebssystem-Absturz) wird im folgenden als die

"konsequenteste Form des bedienerarmen Betriebes"

einer EDVA bezeichnet.

Zeitweilig operatorloser Betrieb einer EDVA

Die weitere Automatisierung der Bedienung einer EDVA führt von einem bedienerarmen Betrieb zum zeitweilig operatorlosen Betrieb.
Der Unterschied zur konsequentesten Form des bedienerarmen Betriebes ist die Möglichkeit des automatischen Wiederanlaufes einer EDVA nach einem Betriebssystem-Absturz, und die dadurch erreichbare größere Systemverfügbarkeit.
Auch hier werden durch technologische Maßnahmen die Zeitabschnitte geschaffen, in denen die Zentraleinheit ausschließlich mit

vormontierten Datenträgern arbeitet.

Vollständig operatorloser Betrieb einer EDVA

Beim vollständig operatorlosen Betrieb einer EDVA wird der - ohne Anwesenheit eines Operators durchgeführte - Betrieb nicht mehr von bedienten Arbeitsabschnitten abgelöst. Durch die Operator wird nur noch eine "Wartung" der EDVA durchgeführt, bei der z. B. die Drucker neu mit Druckpapier beschickt werden.

Der vollständig operatorlose Betrieb setzt eine weitestgehende Automatisierung bzw. Substituierung der manuellen Prozesse bei der Bedienung der EDVA voraus, wie z. B.
- operatorlos betriebene Peripherie,
- qualitativ höheres Niveau der Steuerungsalgorithmen der Betriebssysteme, die keine Eingriffe des Operators mehr erfordern,
- Übertragung manueller Bedienhandlungen der Operator an die Nutzer.

3.2.2. Klassifizierung nach der notwendigen Beaufsichtigung

Eine weitere Klassifizierung des Betriebes einer EDVA erfolgt nach der notwendigen Beaufsichtigung.

Beobachteter Betrieb

Der Betrieb der EDVA erfordert hier eine ständige Aufsicht. Das ist die z. Z. übliche Form des Betriebes einer EDVA in der DDR /35/ /38/.
Um jedoch die Wahrscheinlichkeit der Früherkennung eines Schadensereignisses zu erhöhen, wird oftmals durch technische Sicherheitsmaßnahmen eine zusätzliche automatische Überwachung der EDVA und ihres Umfeldes gewährleistet. Das betrifft in der Regel nur einzelne mögliche Schadensereignisse, mit einem hohen Schadensrisiko, wie z. B. die Früherkennung eines Brandes.

Unbeobachteter Betrieb mit Kontrollen

Werden durch technische Sicherheitseinrichtungen alle wahrscheinlichen Schadensereignisse eines Rechenzentrums überwacht, und wird dabei die gleiche bzw. eine höhere Sicherheit erreicht, wie bei einer Beobachtung durch das Bedienpersonal, dann kann die EDVA auch unbeobachtet betrieben werden.

Um nicht für jedes mögliche Schadensereignis aufwendige technische Überwachungseinrichtungen für den unbeobachteten Betrieb installieren zu müßen, kann folgende Lösung gewählt werden:

(1) Installierung von technischen Überwachungseinrichtungen zur Erkennung von Schadensereignissen, die kurzfristig und mit relativ hoher Wahrscheinlichkeit auftreten können. Z. B.:
- Brand,

- Klimastörung,
- Geräteausfall,
- Wasserschaden,
- Einbruch.

(2) Das Auftreten weniger wahrscheinlicher Schadensereignisse mit größerer Auftritts-Wirkungs-Zeitdauer wird durch regelmäßige Kontrollen, der sonst unbeobachtet betriebenen EDVA überwacht.

Vollständig unbeobachteter Betrieb

Der Betrieb der EDVA erfolgt hier völlig unbeobachtet und nur durch technische Sicherheitseinrichtungen (automatische Feuerlöschanlage, Einbruchsüberwachung u. a.) überwacht.
Eine solche Betriebsweise wird vorerst nur in speziellen Anwendungsfällen möglich sein (z. B. unbedienter und unbeobachteter Betrieb eines Knotenrechners in einem Rechnernetz).
Für Stapel-EDVA ist eine solche Betriebsweise noch nicht wirtschaftlich realisierbar.

4. REALISIERUNG DES UNBEDIENTEN BETRIEBES EINER ESER-EDVA

Die Durchführung des unbedienten Betriebes einer EDVA erfordert eine Automatisierung, Substituierung, sowie die räumliche und zeitliche Verlagerung der bisher notwendigen manuellen Bedienhandlungen.
Problematisch dabei ist, daß EDVA heute - auch international - noch nicht ausreichend für einen unbedienten Betrieb ausgelegt sind /5/ /11/ /12/ /55/.

4.1. Problemstellungen des unbedienten Betriebes

Bei der Realisierung des unbedienten Betriebes einer ESER-EDVA müssen Lösungen folgender fünf zentraler Problemstellungen gefunden werden:

(1) Das Betriebssystem OS/ES unterstützt nicht den unbedienten Betrieb einer ESER-EDVA.
(2) Nach einem Betriebssystem-Absturz kann ein Betriebssystemneustart (IPL) einer ESER-EDVA nur durch einen Bediener erfolgen.
(3) Die Peripherie einer ESER-EDVA ist für die Bedienung durch Operator vorgesehen.
(4) Die Technologie des Rechenzentrums ist i. d. R. auf einen bedienten Betrieb ausgerichtet.
(5) Alle sicherheitstechnischen Überwachungen sind für Mitwirkungen der Operator ausgelegt.

4.2. Lösung der Problemstellungen des unbedienten Betriebes

Im folgenden werden mögliche Lösungen der o. g. Problemstellungen vorgeschlagen. Dabei soll ein zeitweilig operatorloser und unbeobachteter Betrieb einer ESER-EDVA (mit Kontrollen) (vgl. 3.2.) den Vorschlägen zugrunde liegen.

(1) Automatisierung der Konsolbedienung

Während des unbedienten Betriebes muß das Konsolinterface zwischen Betriebssystem und Operator automatisch bedient werden. Hierfür existiert mit dem System SPOOL ab Version 2.0 eine Lösung /23/.

(2) Automatischer Wiederanlauf

Ein Problem des unbedienten Betriebes ist die Gefahr eines Systemabsturzes. Nach einem Absturz des Betriebssystems muß ein automatischer Neustart durchgeführt werden. Ist aufgrund eines schwerwiegenden und nicht automatisch korrigierbaren Fehlers ein Neustart des Systems nicht möglich, muß ein Operator im Bereitschaftsdienst darüber informiert werden.

(3) Arbeit mit der Peripherie während des unbedienten Betriebes

Während des unbedienten Betriebes arbeitet die Zentraleinheit nur mit vormontierten Magnetplatten. Auf die gewohnte Ein- / Ausgabe wird während dieser Zeit verzichtet. /11/ /12/ /23/
Die Jobstromeingabe (z. B. über Lochkarten oder Lochstreifen u. a.) muß vor dem unbedienten Betrieb und die Ergebnisausgabe (Drucklisten, Lochkartenausgabe u. a.) kann erst nach dem unbedienten Betrieb durchgeführt werden.

(4) Technologische Einbindung des unbedienten Betriebes

Für die technologische Einbindung des unbedienten Betriebes in die Rechenzentrums-Organisation ist der Umfang und die konkrete Form des unbedienten Betriebes, sowie die Qualität der bisherigen Rechenzentrums-Technologie bestimmend.
Schwerpunkte sind abhängig vom konkreten Rechenzentrum. /9/ /11/ /12/ /16/ /18/ /21/ /23/ /29/ /32/ /39/ /55/

(5) Sicherheitstechnische Überwachung der unbeobachteten EDVA

Die Ablehnung des unbeobachteten Betriebes einer EDVA erfolgt oftmals aus sicherheitstechnischen Überlegungen /35/.
Soll eine EDVA unbeobachtet betrieben werden, müssen höhere Anforderungen an die Überwachungseinrichtungen gestellt werden, als sie für einen beobachteten Betrieb notwendig sind.

Bei der Erarbeitung eines Sicherkeitskonzeptes für den unbeobachteten Betrieb sind vorrangig die Risiken zu beachten, die bisher

durch die Anwesenheit von Kontrollpersonen ausgeschlossen wurden. Das betrifft vor allem die Risiken:
- Brand,
- Klimastörung,
- Wasserschaden,
- Einbruch (Spionage, Rowdytum u. a.).

Außerdem sind die örtlichen Gegebenheiten und das Nutzungsprofil während des unbeobachteten Betriebes zu berücksichtigen. Das betrifft vor allem die Gegebenheiten:
- Lage des Rechenzentrums und dadurch ausgeschlossene bzw. daraus resultierende Risiken;
- bereits vorhandene Sicherheitsvorkehrungen und -einrichtungen;
- Vertraulichkeitsgrad der bearbeiteten Daten;
- Möglichkeit von regelmäßigen Kontrollgängen während des unbeobachteten Betriebes der EDVA.

/2/ /11/ /12/ /13/ /20/ /25/ /35/

5. ENTWICKLUNGSTENDENZEN BEIM BEDIENERARMEN BZW. UNBEDIENTEN BETRIEB

5.1. Realisierte Lösungen

Im folgenden werden zwei Fälle vorgestellt, bei denen die konsequenteste Form des bedienerarmen Betriebes bzw. der unbediente Betrieb einer EDVA realisiert wird.

5.1.1. Konsequenteste Form des bedienerarmen Betriebes (unbeobachtet mit Kontrollen)

In Zentrum für Datenverarbeitung der Universität Tübingen arbeiten zwei zentrale EDVA während der Nachtschicht und während des Wochenendes ohne Operator - d. h. unbeobachtet mit Kontrollen, in der konsequentesten Form des bedienerarmen Betriebes. /12/

Die hohe Verfügbarkeit der EDVA und sicherheitstechnische Überwachungseinrichtungen (Klimaüberwachung, Brandmelder mit automatischer Feuerlöschanlage und automatischer Alarmierung der Feuerwehr) ermöglichen den unbeobachteten Betrieb mit Kontrollen.
Dabei werden zur Kontrolle der EDVA auch die Nutzer hinzugezogen, die durch eine Glasscheibe aus der Expreßstation in die Rechnerräume sehen können. Sie sind aufgefordert, bei erkannten Unregelmäßigkeiten telefonisch einen diensthabenden Operator zu benachrichtigen.
Man verläßt sich hier auf die fast lückenlose Kontrolle der EDVA durch die Benutzer der Expreßstation auch in den Nachtstunden und am Wochenende /12, S. 17/.
Bei einem Betriebssystem-Absturz wird ein längerer Stillstand der EDVA akzeptiert, da ein automatischer Wiederanlauf der EDVA nicht vorgesehen ist. Jedoch tritt aufgrund einer hohen Systemverfügbarkeit nur selten ein Ausfall der EDVA ein /12, S. 21/.

5.1.2. Zeitweilig bis vollständig operatorloser Betrieb (unbeobachtet mit Kontrollen)

Eine zukunftsweisende Form des unbedienten Betriebes einer EDVA wird in /17/ aus japanischen Hochschul-Rechenzentren berichtet. Durch

"- Übertragung von Arbeiten, die bisher von Operateuren ausgeführt wurden, auf die Benutzer
- Automation im Maschinensaal durch Wegfall bisher manuell zu verrichtender Arbeiten
- Automatisierung im Steuerungsablauf der DV-Systeme"
/17, S. 137/

wurde in einigen japanischen Hochschul-Rechenzentren ein vollständig operatorloser Betrieb realisiert.

Über den Umfang der notwendigen Beobachtung der EDVA wurden in /17/ keine Aussagen gemacht.

Als entscheidende Voraussetzung für die operatorlose Betriebsweise in den beschriebenen japanischen Hochschul-Rechenzentren wird die großzügige Geräteausstattung genannt, vor allem hinsichtlich

- einer großen Direktzugriffs-Speicherkapazität aus unbedient betriebenen Plattengeräten bzw. Massenspeichersystemen für die
  * Zwischenspeicherung des Jobstromes und aller Druckausgabedateien bis zu ihrem Abruf durch die Nutzer (Spooling),
  * für die Aufnahme aller Nutzerdateien /17, S. 138, 140/.

Weiterhin steht in Expreßstationen der beschriebenen japanischen Hochschul-Rechenzentren eine große Zahl von Lochkartenlesern bzw. Bildschirmterminals für die Jobstromeingabe, sowie von Magnetbandgeräten (die vom Benutzer selbst bedient werden) zur Verfügung /17, S. 138, 139/.

Die Überlegungen zur Realisierung eines bedienerarmen bzw. unbedienten Betriebes in Rechenzentren der DDR müssen von anderen, als den in /17/ genannten, Hardware-Voraussetzungen ausgehen.

5.2. Internationale Entwicklungstendenzen

In (2.) wurden bereits internationale Entwicklungstendenzen beschrieben, die zukünftig eine zunehmende Automatisierung der Rechenzentrums-Funktionen - auch der Maschinenbedienung - ermöglichen und erfordern werden.
Außerdem läßt sich international auch ein großes Interesse der Rechenzentrum-Betreiber an einem bedienerarmen bzw. zeitweilig operatorlosen Betrieb der genutzten EDVA erkennen. So wird in /5/, /12/, /39/, /44/ von künftigen EDVA eine bessere Hardware- und Software-Unterstützung der unbedienten Betriebsweise gefordert. (siehe auch 2. Punkt 6)

Erste Tendenzen hierzu sind bei den Computerherstellern bereits zu erkennen.

So ist z. B. das neue UNIVAC 1100/90 - System mit einem Service-Prozessor ausgerüstet, der als dedizierter Minicomputer die folgenden Aufgaben hat (/1, S. 63 - 64/):

- Hardwareüberwachung der zentralen EDVA (mit der Fähigkeit der Wiederherstellung oder Abschaltung von fehlerhaften Systemkomponenten)
- Initialisierung des gesamten Systems (Initialisierung der Einzelkomponenten und IPL-Ladevorgang)
- Unterstützung von Wartungsaktivitäten.

In /15/ wird z. B. über Aktivitäten des Computerherstellers IBM berichtet, die zum Ziel haben, alle Rechner eines Rechnerverbundes, ausschließlich von einer zentralen Stelle aus steuern zu können. Dazu wurde ein Überwachungsrechner implementiert, "indem die Systemmeldungen aller betriebenen Rechner konzentriert und interpretiert werden und der alle Kommandos für die einzelnen Rechner weitervermittelt ..." /15, S. 197 - 198/.

6. LITERATURVERZEICHNIS

/1/ Adler, Hans-Joachim
Architektur von Großrechnern - Beispiel SPERRY UNIVAC 1100/90
in: Handbuch der modernen Datenverarbeitung HDM; Forkel-Verlag Stuttgart-Wiesbaden 19(1982) Heft 107 September, S. 59 - 66

/2/ Allianz Versicherungs-AG (Hrsg.)
Handbuch der Schadensverhütung
3.1. Elektronische Datenverarbeitung (EDV-Anlagen)
in: Handbuch der Schadensverhütung 2., erweiterte und überarbeitete Auflage, Allianz Versicherungs-AG München, Berlin(West) 1976, S. 121 - 139

/3/ Bergholz, Gerhard
Leistungsbewertung im Rechenbetrieb
in: Rechentechnik Datenverarbeitung Berlin 21(1984) Heft 1, S. 10 - 13

/4/ Cuttle, G.; Robinson, P. B.
Executive programms and operating systems
Mac Donald London and American Elsevier Inc. New York 1970

/5/ Dirlewanger, W.
Integration von Funktionen betrieblicher Rechenzentren in den technisch - wissenschaftlichen Rechenbetrieb
in: Organisation und Betrieb von Rechenzentren, Fachgespräch der GI, Erlangen, 12./13. März 1981, Informatik Fachberichte Band 46, Brauer, W. (Hrsg.) Springer-Verlag Berlin(West), Heidelberg, New York 1981, S. 116 - 129

/6/ Dückerkop, Klaus
Ablaufplanung im RZ der ARAL AG - Anwendererfahrungen mit OPC
in: Ablaufplanung und Arbeitsvorbereitung im Rechenzentrum Fachseminar Universität Köln, 7./8. Februar 1980 im Helfta-Seminar, Seibt, Dietrich (Hrsg.), Köln 1980, 3. Kapitel

/7/ Faltenbach, Hans
Methoden und Verfahren zur Steuerung von Großrechenzentren Teil 1 und Teil 2
in: IBM-Nachrichten Stuttgart 28(1978) Heft 243, S. 356 - 361; 29(1979) Heft 244, S. 36 - 44

/8/ Gaffal, Franz
Datenverarbeitung im Hochschulbereich der USA - Stand und Entwicklungstendenzen
Informatik Fachberichte Band 26, Springer-Verlag Berlin(West), Heidelberg, New York 1980

/9/ Graef, Martin (Hrsg.)
Benutzerleitfaden - Zentrum für Datenverarbeitung der Universität Tübingen Ausgabe: Oktober 1981
Universität Tübingen, Zentrum für Datenverarbeitung, Tübingen 1981

/10/ Graef, Martin
Das Rechenzentrum-Personal der Zukunft
in: Betrieb von DV-Systemen in der Zukunft, 5. GI-Fachgespräche über Rechenzentren, Tübingen, 17./18. März 1983; Graef, Martin (Hrsg.), Informatik Fachberichte Band 69; Springer-Verlag Berlin(West), Heidelberg u. a. 1983, S. 334 - 341

/11/ Graef, Martin; Greiller, Reinald
Organisation und Betrieb eines Rechenzentrums
Schriftenreihe Integrierte Datenverarbeitung in der Praxis Band 13, 2., überarbeitete Auflage; Forkel-Verlag Stuttgart, Wiesbaden 1982

/12/ Graef, Martin; Maier, Edgar
Erfahrungen über operateurlose RZ-Schichten
in: Das Rechenzentrum München, Wien 6(1983) Heft 1, S. 15 - 23

/13/ Höfling, Bernd; Zwingert, Gerd
Sicherheit im Rechenzentrum
in: Handbuch der modernen Datenverarbeitung HDM; Forkel-Verlag Wiesbaden, Stuttgart 19(1982) Oktober Lieferung 103a 9/4/37, Blatt 1 - 24

/14/ Hofmann, Hans-Christoph
Untersuchungen der Dialogwürdigkeit für die Kontrolle und Lenkung der Produktion am Beispiel des VEB DVZ Halle
Diplomarbeit, Ingenieurhochschule Dresden, Sektion Informationsverarbeitung; Dresden 1980

/15/ Hultzsch, Hagen
SEAS, SHARE EUROPEAN ASSOCIATION Bericht vom Spring Meeting 1982 in Noordwijkerhout/Holland
in: Das Rechenzentrum München, Wien 5(1982) Heft 3, S. 195 - 198

/16/ Jasper, Erich
Arbeitsvorbereitung als Möglichkeit zur Steigerung der Effizienz des Hochschulrechenzentrums
in: Organisation von Rechenzentren (Workshop der Gesellschaft für Informatik, Göttingen 11./12. Oktober 1977) Informatik Fachberichte Band 15 Springer-Verlag Berlin(West), Heidelberg, New York 1978, S. 13 - 22

/17/ Jasper, Erich
Organisation und Betrieb japanischer Hochschulrechenzentren Teilaspekt: Operateurloser Betrieb
in: Das Rechenzentrum München, Wien 5(1982) Heft 3, S. 137 - 141

/18/ Jochum, Hans-Günter
Probleme und Realisierung eines unbedienten Rechenzentrums-Betriebes
in: Betrieb von DV-Systemen in der Zukunft, 5. GI-Fachgespräche über Rechenzentren, Tübingen, 17./18. März 1983; Graef, Martin (Hrsg.), Informatik Fachberichte Band 69 Springer-Verlag Berlin(West), Heidelberg u. a. 1983, S. 51 - 60

/19/ Kießling, Helge
Methoden und Techniken der Organisation eines Rechenzentrums
in: Datascope Sulzbach/Taunus 10(1979) Heft 29, S. 9 - 22

/20/ Knauer, B.
Ein Überwachungssystem für den unbeaufsichtigten Betrieb der TR 440
in: Angewandte Informatik Braunschweig 20(1978) Heft 12, S. 526 - 528

/21/ Knop, Jan; Stichtenoth, Horst; Haverkamp, Wilhelm
Organisation der automatisierten Datenverarbeitung an Hochschulen in Nordrhein-Westfalen
in: data report Berlin(West), München 15(1980) Heft 4, S, 19 - 24

/22/ Kooi, Horst
Die Eignungsuntersuchung von Rechenzentrums-Personal
in: Das Rechenzentrum München, Wien 6(1983) Heft 1, S. 46 - 50

/23/ Kroß, Günter
SPOOL 2.0 - System zur Unterstützung eines zeitweise bedienungsunabhängigen Betriebes an ESER-Rechnern
in: Berliner Informatik-Tage bit'82, 28. Juni bis 2. Juli 1982, Tagungsbericht, Humboldt-Universität zu Berlin, Organisations- und Rechenzentrum; Berlin 1982, S. 485 - 490

/24/ Löffler, H.
Ziele, Methoden und Probleme der Analyse komplexer Informationssysteme
in: Vorträge zur Analyse in EDVA und IV-Systemen, Problemseminar des Weiterbildungszentrums für Mathematische Kybernetik und Rechentechnik / Informationsverarbeitung; Technische Universität Dresden; Heft 53/80, S. 3 - 13

/25/ Manuel, Tom
McAuto: the good life for computers
in: Electronics New York 54(1981) Heft 11, S. 98 - 99

/26/ Meier, Hermann Walter
Rechnernetz DELTA Konzept, erste Einsatzvariante und Dienste
in: Rechentechnik Datenverarbeitung Berlin 20(1983) Heft 6, S. 6 - 11

/27/ Meißler, Hans-Günter
Gutachten zum Pilotsystem bedienerarme Rechnernutzung - SPOOL 3 -
Martin-Luther Universität Halle-Wittenberg; Organisations- und Rechenzentrum; Halle-Wittenberg 1983

/28/ Meißler, Hans-Günter
SPOOLRJE* - eine Fernverarbeitungskomponente des SPOOL-Systems
in: Berliner Informatik-Tage bit'82, 28. Juni - 2. Juli 1982, Tagungsbericht, Humboldt-Universität zu Berlin, Organisations- und Rechenzentrum; Berlin 1982, S. 491 - 497

/29/ Mertens, B.; Hoßfeld, F.
Jobsteuerung und Überwachung in einem wissenschaftlichen Rechenzentrum
in: GI - 9. Jahrestagung Bonn 1. - 5. Oktober 1979, Informatik Fachberichte Band 19; Böhling, K. H.; Spies, P. P. (Hrsg.), Springer-Verlag Berlin(West), Heidelberg, New York 1979, S. 545 - 555

/30/ Muchsel, Reginald
Kontingentierung und Ressourcenverteilung an Hochschulrechenzentren
in: Das Rechenzentrum München, Wien 6(1983) Heft 3, S. 172 - 181

/31/ Näther, Bernd
Ein Beitrag zur Erhöhung der Effektivität der Technologie und Organisation von Rechenzentren mit ESER-Rechentechnik
Dissertation, Ingenieurhochschule Dresden 1981

/32/ O'Neill, Paul
Unattended Operations Using OPC
in: Das Rechenzentrum München, Wien 4(1981) Heft 2, S. 83 - 91

/33/ Pfeiffer, Peter
Automatisierung der Technologie - Mittel zur Rationalisierung
in: Rechentechnik Datenverarbeitung Berlin 20(1983) Heft 2, S. 1

/34/ Piens, Conrad
Effektivitätsuntersuchung für das System SPOOL 3
in: Berliner Informatik-Tage bit'82, 28. Juni bis 2. Juli 1982 Tagungsbericht, Humboldt-Universität zu Berlin, Organisations- und Rechenzentrum; Berlin 1982, S. 498 - 502

/35/ Pinus, Rudolf; Mathauser, Pavel
Brandschutztechnische Maßnahmen in Rechenstationen
in: Brandschutz Explosionsschutz, Aus Forschung und Praxis 4 Staatsverlag der Deutschen Demokratischen Republik; Berlin 1980, S. 143 - 189

/36/ Pralle, H.
Auswirkungen der organisatorischen und technischen Entwicklungstendenzen auf Großrechenzentren
in: Organisation und Betrieb von Rechenzentren, Fachgespräch der GI, Erlangen, 12./13. März 1981, Informatik Fachberichte Band 46; Springer-Verlag Berlin(West), Heidelberg, New York 1981, S. 166 - 168

/37/ Rackles, R.
Der Betriebsablauf in einem Konzern-Rechenzentrum
in: Betrieb von Rechenzentren (Workshop der Gesellschaft für Informatik Karlsruhe, 23. - 24. September 1975); Schreiner, A. (Hrsg.); Informatik Fachberichte Band 2; Springer-Verlag Berlin(West), Heidelberg, New York 1976, S. 73 - 104

/38/ Richter, Rudi
Produktionstechnologie im Rechenzentrum
Schriftenreihe Informationsverarbeitung; Verlag die Wirtschaft; Berlin 1979

/39/ Rippel, G.; Jasper, E.; Kneip, W.
Massenspeichersysteme - ein erster Schritt auf dem Weg zum operatorlosen Betrieb
in: Organisation und Betrieb von Rechenzentren, Fachgespräch der GI, Erlangen, 12./13. März 1981, Brauer, W. (Hrsg.); Informatik Fachberichte Band 46; Springer-Verlag Berlin(West), Heidelberg, New York 1981, S. 176 - 187

/40/ Sack, Kurt
Zentral, dezentral oder kombiniert? Eine Betrachtung zum Inhalt und zu Entwicklungstendenzen von Organisationsformen der Datenverarbeitung
in: Rechentechnik Datenverarbeitung Berlin 17(1980) Heft 10, S. 17 - 18

/41/ Schiller, Hans-Jürgen
Ein Beitrag zur Erhöhung des Jobdurchsatzes bei Multiprogramm Betrieb für mittlere und große EDVA mit einem Rechenwerk
Dissertation, Technische Universität Dresden, Fakultät für Elektrotechnik und Elektronik; Dresden 1977

/42/ Schmitz, P.
Zukünftige Nutzungsform der ADV im Hochschulbereich
in: Angewandte Informatik Braunschweig 20(1978) Heft 5, S. 194 - 202

/43/ Schmitz, Paul; Krekel, Dietrich
Aufgabenorientiertes Modell zur Nutzung der automatisierten Datenverarbeitung im Hochschulbereich (Teil 1 und Teil 2)
in: Das Rechenzentrum München, Wien 5(1982) Heft 2, S. 110 - 125; 5(1982) Heft 3, S. 142 - 158

/44/ Seibt, Dietrich
Software zur Unterstützung von Rechenzentrumsfunktionen
in: Angewandte Informatik Braunschweig 24(1982) Heft 2, S. 104 - 114

/45/ Seibt, Dietrich; Langen, Bernhard; Barthel, Andreas; Kirchhöfer, Klaus
Erhebung zur statistischen Planung der Dienstleistungen von Service-Rechenzentren
Teil A: Auswertung der postversandten Fragebögen
BIFOA-Arbeitspapier 80AP8 (Zugleich Bericht Nr. 1 des Projektes SEDIS); Köln 1980

/46/ Siefarth, Gerhard
Neue Maßstäbe für eine moderne Leitungstätigkeit
in: Rechentechnik Datenverarbeitung Berlin 16(1979) Heft 3, S. 14 - 17

/47/ Sitta, Heinz-Günter
Untersuchung zur Bestimmung und Analyse von Einflußgrößen bei Störungen im Produktionsprozeß am Beispiel der ESER-Anlage im VEB DVZ Halle
Diplomarbeit, Martin-Luther-Universität Halle-Wittenberg, Sektion Wirtschaftswissenschaften; Halle 1980

/48/ Stuchlik, Franz; Werler, Karl-Heinz
Zur Entwicklung neuer Technologien geistiger Arbeit
in: Wissenschaftliche Zeitschrift der Technischen Hochschule "Otto von Guericke" Magdeburg 27(1983) Heft 3, S. 93 - 98

/49/ Wähner, Gerd
Grundsätze und Methoden zur rechnergestützten Planung und Analyse der Auslastung von EDVA des ESER im kurzfristigen Planungszeitraum
Inaugural-Dissertation, Hochschule für Ökonomie "Bruno Leuschner"; Berlin 1978

/50/ Wähner, Gerd
Organisation der Bedienung von EDVA
in: Rechentechnik Datenverarbeitung Berlin 19(1982) Heft 1, S. 31 - 33

/51/ Wall, Dieter
Der organisatorische Gestaltungsspielraum des Rechenzentrums bei den Betriebsarten Stapel- und Dialogverarbeitung
in: Das Rechenzentrum München, Wien 6(1983) Heft 1, S. 24 - 36

/52/ Winks, Hans
SPOOL - ein System zur Automatisierung von Produktionsprozessen an ESER-Rechnern; Entwicklungsstand und Entwicklungsrichtungen
in: Berliner Informatik-Tage bit'82, 28. Juni bis 2. Juli 1982, Tagungsbericht, Humboldt-Universität zu Berlin, Organisations- und Rechenzentrum; Berlin 1982, S. 40 - 57

/53/ Wolf, F.
Das Rechenzeit-Abrechnungsverfahren an der Universität Erlangen-Nürnberg
in: Abrechnung von Rechenzentrums-Dienstleistungen; Mertens, Peter (Hrsg.), Applied Computer Science, Berichte zur praktischen Informatik 12, Carl Hanser Verlag München, Wien 1978, S. 133 - 144

/54/ o. N.
Computer schaltet sich selbst ein
in: Siemens data report Berlin, München 18(1983) Heft 6, S. 39

/55/ o. N.
In eigener Sache
in: Login - Informationsschrift des Hochschulrechenzentrums Gießen 2(1981) Heft 2, S. 19 - 20

Verfasser: Prof. Dr. rer. nat. Franz Stuchlik
Dipl.-Ing. Werner Winzerling
Technische Hochschule "Otto von Guericke"
Sektion Rechentechnik / Datenverarbeitung
3010 Magdeburg
Boleslaw-Bierut-Platz-5
Tel: Mgb. 592805

Sommerfeld, Michael

Simulation der Ressourcenauslastung und des Jobdurchsatzes eines Rechnersystems unter dem Betriebssystem OS/MVT und dem Preschedulingsystem SPOOL

1. Motivation der Fragestellung

Intuitive Festlegungen der Hardwarestruktur, der Betriebssystemfunktionen und der Produktionstechnologien reichen nicht mehr aus, um die Leistungsmöglichkeiten heutiger und zukünftiger Rechnersysteme effektiv nutzen zu können. Deshalb gewinnt die Anwendung von Realisierungen mathematischer Modellklassen, auf die sich wesentliche Eigenschaften der Rechnersysteme abbilden lassen, zunehmend an Bedeutung. Beispiele für Realisierungen solcher Modellklassen sind die Simulationssysteme SIMDIS, GASPIV, SIMKOM-F und -S, Realisierungen der S-, E- und Pro-Netze und das in Entwicklung befindliche System MARS. Etwas eingeschränktere Modellierungsmöglichkeiten bieten gegenwärtig noch Anwendungen der Bedienungstheorie und der hybriden Simulation (/1/, /2/).

Am ORZ der Humboldt-Universität wird das System SPOOL entwickelt, das durch die Beseitigung verschiedener Schwachstellen des OS/MVT die bessere Bedienbarkeit und effektivere Auslastung entsprechender Rechnersysteme ermöglicht. Die geeignete Nutzung von SPOOL hängt jedoch von vielen Faktoren ab. Solche wären z. B. geeignete Hardware des Rechnersystems, angepaßte Generierung des OS/MVT und des Systems SPOOL wie auch Festlegung einer günstigen Produktionstechnologie. Die Simulation der Ressourcenauslastung eines Rechnersystems unter Berücksichtigung derartiger Faktoren wird die Bestimmung von Engpässen und teilweise deren zielgerichtete Entschärfung ermöglichen. Die Simulation bietet sich zur Lösung dieser Aufgabe an, da Ergebnisse handhabbarer bedienungstheoretischer Modelle, wegen des nicht konstanten Jobprofils und der prioritätsbezogenen Vergabe der CPU, anzuzweifeln sind (/1/).

Zur Modellierung wurde das System MARS (Modellierung und Analyse von Rechner- und Steuersystemen), das gegenwärtig am Bereich Informationsverarbeitung der Sektion Mathematik der HUB imple-

mentiert wird, ausgewählt.

2. Erläuterung des Modells

2.1. MARS /3/

Das System MARS ermöglicht die Auswertung von M-Netzen (modifizierte zeitbewertete Petri-Netze). Von Petri-Netzen unterscheiden sich diese durch attributierte Marken, Feuerzeiten, konfliktlösende Transitionsprozeduren, markenattributändernde Transitionsprozeduren, komplexere Struktur der Transitionen, durch Realisierung der Plätze als FIFO-Warteschlangen und die Nutzungsmöglichkeit lokaler und globaler Variablen.

Die komplexere Struktur der Transitionen besteht in der Möglichkeit der Gruppenbildung von Eingangspfaden bzw. Ausgangspfaden, die durch einen die Pfade verbindenden Querstrich dargestellt wird. Von den Gruppen ist jeweils genau eine der Eingangsgruppen und eine der Ausgangsgruppen an jedem Feuern beteiligt. Pfade, die nicht durch Querstriche gekennzeichnet sind, sind immer am Feuern beteiligt. Eine Transition kann erst dann wieder neu aktiviert werden, wenn die Feuerzeit der letzten Aktivierung abgelaufen ist und mindestens eine Feuermöglichkeit besteht.

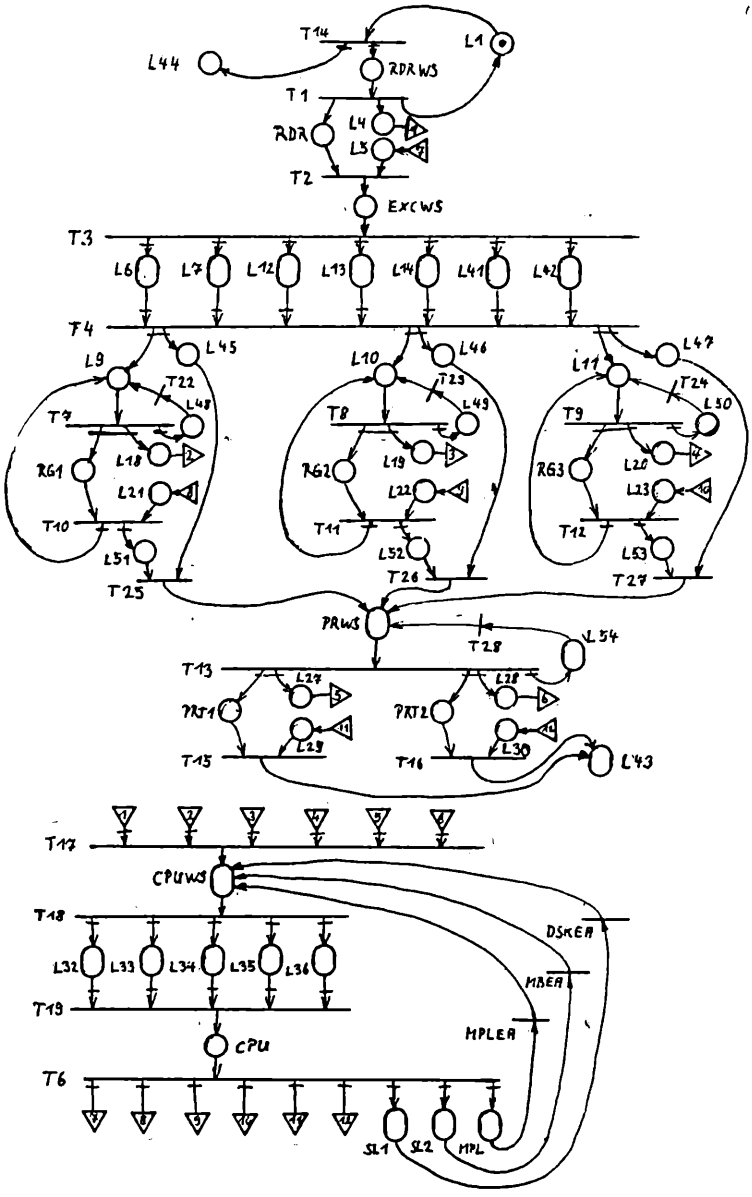
Das System MARS wird vollständig in PASCAL implementiert. Gegenwärtig ist die Modellierung durch das Fehlen eines Precompilers und die Wiedergabe der Systemzeit durch eine Variable des Typs SHORTINT erschwert.

2.2. Eingabedaten des Modells

Ein aus SMF-Daten des Produktionsbetriebs erzeugter abstrakter Jobstapel stellt die Eingabe des Modells dar. Jeder abstrakte Job wird während der Abarbeitung des M-Netzes einer Marke zugewiesen, mit der er das Netz durchläuft.

Ein abstrakter Job beinhaltet:

- Readerzeit (Verweilzeit des Jobs im SPOOL- und im OS-Reader sowie den SPOOL-Checkprozessoren)
- Stepanzahl (auf maximal 5 Steps begrenzt - Verbrauchswerte für weitere Steps fortlaufend auf die ersten 5 addiert)


T14
L1
L44
RDRWS
T1
RDR
L4
L5
T2
EXCWS
T3
L6
L7
L12
L13
L14
L41
L42
T4
L45
L46
L47
L9
L10
L11
T22
T23
T24
L48
L49
L50
T7
T8
T9
RG1
L18
L21
RG2
L19
L22
RG3
L20
L23
T10
T11
T12
L51
L52
L53
T25
T26
T27
PRWS
T28
L54
T13
L27
L28
PRT1
PRT2
L29
L30
T15
T16
L43
T17
CPUWS
T18
DSKEA
L32
L33
L34
L35
L36
MBEA
T19
MPLEA
CPU
T6
SL1
SL2
MPL

- Druckzeilenanzahl (Formate nicht unterschieden)
- Kartenanzahl (geschätzter Wert, da SMF-Daten über die Aktivitäten des Systems SPOOL nicht aufgezeichnet werden)
- Jobklassen (durch maximalen HS-Bedarf bestimmt)
- Jobnummer

Stepbezogene Angaben:
- Hauptspeicherbedarf
- CPU-Zeit
- Anzahl der E/A für
  - Plattenstrecke (Selektorkanal 1)
  - Bandstrecke (Selektorkanal 2)
  - Drucker, Kartenleser (Multiplexkanal)
- Anzahl angesprochener Dateien
- Anzahl angesprochener Magnetbänder
- Stepinitialisierungszeit

2.3. Modellstruktur

Das abgebildete Netz des Rechnersystemmodells besteht aus einem Readerprozeß, einer CPU, 2 Selektorkanälen, einem Multiplexkanal, 3 Nutzerprogrammregionen, 2 Druckprozessen, einem Jobstrom, der aus bis zu 7 verschiedenen Jobklassen bestehen kann, und 5 CPU-Auswahlprioritäten, von denen die beiden höchsten für SPOOL und OS/MVT reserviert sind.
Die Transition T14 und die Plätze L1 und RDRWS dienen lediglich der kontinuierlichen Versorgung des Readerprozesses mit einer zuvor festgelegten Anzahl von Marken. Deshalb ist der Readerprozeß nur in einer Anfangsphase aktiv.
Das Teilnetz, bestehend aus den Transitionen T17, T18, T19, T6, MPLEA, MBEA und DSKEA sowie den von ihnen eingeschlossenen Plätzen modelliert die Vergabe der CPU und der Kanäle an die System- und die Nutzeraufgaben. Dabei stellen die Gesamtfeuerzeiten der Transitionen T6, MPLEA, MBEA bzw. DSKEA die Auslastungen der CPU, des Multiplexkanals, des Selektorkanals für die Magnetbandstrecke bzw. des Selektorkanals für die Plattenstrecke dar. Die Wartezeiten der Aufgaben lassen sich aus den Auswertungsgrößen Belegungsdauer und durchschnittliche Belegungszahl der Plätze L32 bis L36, SL1, SL2 und MPL ermitteln.

Der noch nicht genauer erklärte Teil des Netzes realisiert die Auswahl der Jobs bzw. Jobsteps für die einzelnen Phasen der Abarbeitung. So wird mit dem Feuern von T1 der Readerprozeß aktiviert, der beim Feuern von T2 für den entsprechenden JOB beendet ist. Die mit der Transition T3 verbundenen Plätze stellen die nach Klassen organisierte Ausführungswarteschlange dar. T3 stellt dabei die Ordnung nach Jobklassen her.
T4 versorgt die 3 Initiatoren für Nutzeraufgaben, deren Steuerung zwischen T4 und dem Platz PRWS dargestellt ist, mit Marken. Nehmen wir an, daß T4 eine Marke gedoppelt und auf die Plätze L9 und L45 gebracht hat. Der Platz L45 zeigt an, daß der entsprechende Initiator belegt ist. Zur selben Zeit wird damit T7 feuerbereit und bewegt die Marke von L9 entweder auf L48 (falls nicht genügend zusammenhängender Hauptspeicherplatz für die Stepausführung verfügbar ist) oder doppelt sie und bewegt sie auf die Plätze RG1 und L18. Kommt die Marke von L18 schließlich aus dem CPU-E/A-Zyklus auf den Platz L21, so wird T10 feuerbereit. T10 bewegt die Marke auf L51, falls der letzte Step abgearbeitet worden war oder sonst auf L9, womit eine neue Stepinitiierung beginnt. Die Transition T22 verzögert eine Marke, die auf L48 gelangt ist, für eine gewisse Zeit, um ihr danach erneut den Test auf Verfügbarkeit von Hauptspeicherplatz zu ermöglichen. Mit dem Feuern der Transition T25 wird der Initiator wieder verfügbar, da L9 und L45 keine Marken mehr enthalten, wie auch das ganze Teilnetz.
T13 verteilt die Druckausgaben der Jobs auf die beiden Druckerprozesse bzw. überführt sie in einen Wartezustand, falls kein Druckprozeß verfügbar ist.

3. Validierung des Modells

Zur Validierung des Modells wurden SMF-Daten, die am modellierten Rechnersystem aufgezeichnet wurden, zur Erzeugung von praxisnahen Jobströmen und parallel zur Bestimmung der realen Ressourcenauslastung und des Durchsatzverhaltens

ausgewertet. Um Verfälschungen der Ergebnisse zu vermeiden, wurden Zeitabschnitte betrachtet, an deren Grenzen kein Job im System aktiv war und innerhalb derer ständig mindestens ein Job ausgeführt wurde. Außerdem werden verschiedene Phasen des Stapelbetriebes wie Expreßbetrieb, bedienarmer Betrieb und Nachtschichtbetrieb untersucht werden. Wenn durch Modifikationen am Modell die Simulationsergebnisse hinreichend genau denen entsprechen, die aus den SMF-Daten ermittelt wurden, können die eigentlichen Ziele der Modellierung in Angriff genommen werden.
Voraussagen über die Ergebnisse einer Regionnormierung, modifizierter Jobklasseneinteilungen, veränderter SPOOL-Generierungsparameter, von Hardwareerweiterungen und Hardwareänderungen zu erzielen, werden Ziele der weiteren Modellierung sein.

4. Grenzen der Ausdrucksfähigkeit und Effizienzprobleme

Nicht alle wesentlichen Einflußgrößen auf die Ressourcenauslastung und den Jobdurchsatz können so exakt wie die Summe der CPU-Zeiten und die der Kanalbelegungszeiten der Jobs im Modell berücksichtigt werden. Solche Einflußgrößen sind beispielsweise die innere Parallelität der Jobs, das reale Zeitverhalten der Jobs im Ausführzyklus, die Bedienaktionen in der Produktion, die die SMF-Daten lieferte, Beschränkung auf die Widerspiegelung relativ kurzer Produktionsphasen und Vergröberung des Ausführungszyklus durch Zusammenfassung mehrerer E/A zu einer einzigen. Die letzten beiden Einschränkungen sind nötig, um die Rechnerbelastung durch die Simulation in vertretbaren Grenzen zu halten. Zur Berücksichtigung der anderen Größen können nur Erfahrungswerte herangezogen werden. Eine erhöhte innere Parallelität von Jobs führt zu einer höheren Ressourcenauslastung und damit bei gleichen quantitativen Lasten für die Ressourcen zu erhöhtem Jobdurchsatz. Somit könnte der relative Unterschied zwischen der realen Ausführung nur eines Jobs und dessen

simulierter Ausführung weitgehend als Maß seiner inneren Parallelität angesehen werden.
Bei der parallelen Ausführung mehrerer Aufgaben hat das Zeitverhalten und die Priorität jeder einzelnen Einfluß auf die Ressourcenauslastung. Durch geeignete Wahl von Zufallszahlengeneratoren zur Festlegung der Belegungsdauer der einzelnen Ressourcen wird es möglich sein, eine gute Näherung zu erreichen. Doch wird sich auf jeden Fall die Tatsache, daß das Modell der CPU gegenwärtig noch nicht unterbrechbar ist, negativ auf den Auslastungsgrad der Ressourcen im Multiprogrammbetrieb auswirken. Erst wenn diese Unterbrechbarkeit realisiert ist, werden sinnvolle Ergebnisse der Simulation einer dynamischen Prioritätensteuerung zu erwarten sein. Bei der Auswertung der Simulationsergebnisse kann dieser Mangel gegenwärtig nur durch einen Korrekturfaktor berücksichtigt werden. Das beschriebene Modell besteht aus 82 Steuerkarten zur Netzgenerierung und etwa 700 PASCAL-Zeilen. Dieser Aufwand ließe sich um 20 % verringern, wenn die M-Netze nicht nur FIFO-Warteschlangen unterstützen würden. Auch ist absehbar, daß die Realisierung einer durch mehrere Prioritätsklassen unterbrechbaren CPU zu einem aufwendigen Teilnetz führt, womit das Modell weiter aufgebläht und die Effizienz stark herabgesetzt würde. Durch Einbau eines geeigneten analytischen Modells für den CPU-E/A-Zyklus könnte das Effizienzproblem gelöst werden. Ein solcher Einbau ist aber im Konzept des Systems MARS nicht vorgesehen und deshalb eher in einem hybriden oder kombinierten Simulationssystem erreichbar.

<u>Literaturverweise:</u>

/1/ Carl, D. "Rechnergestützte Verhaltensmodellierung und -bewertung komplexer Rechnersysteme", Studie, ZfR-Information, AdW der DDR, ZfR-84.

/2/ Bergholz, G., Mrosko, G. "Simulationsmodell der Stapelverarbeitung im Multiprogrammbetrieb" Humboldt-Universität zu Berlin, Tagungsbericht der Berliner Informatik-Tage 1982

/3/ Grützner, R. "Konzeptionelle Grundlagen der Modellierung und Simulation von Software auf der Basis modifizierter Petri-Netze", AdW der DDR, ZfR-Information, ZfR - 84.06, 1984

Dipl.-Math. Michael Sommerfeld
Humboldt-Universität zu Berlin
Organisations- und Rechenzentrum
1086 Berlin, Unter den Linden 6

Winke, Hans

Möglichkeiten portabler Basissoftware

Mit dem UNIX-System ist ein Betriebssystem weit verbreitet, das für 16-Bit-Prozessoren implementiert wurde, aber inzwischen auch auf 32-Bit-Prozessoren übertragen wurde. Weniger bekannt ist das THOTH-System, das ebenfalls für 16-Bit-Prozessoren geschaffen wurde und inzwischen auf 6 verschiedenen Mikrorechnern angewendet wird.

Grundlage dieser portablen Betriebssysteme sind Sprachentwicklungen (C und ZED), die besonders auf Bedürfnisse der Systemprogrammierung abgestimmt sind, aber als höhere Programmiersprache hinreichende Distanz zu Maschinenabhängigkeiten aufweisen. Die Verfügbarkeit von Compilern für diese Sprachen auf geeigneten, in der DDR verfügbaren Rechnern gibt die Möglichkeit der Übertragung dieser Systeme. UNIX und THOTH sind Modellfälle für portable Basissoftware. Sie werden im Rahmen des Beitrages kurz charakterisiert und es werden die Architekturprinzipien erläutert.

Dipl.-Math. H. Winke
Humboldt-Universität Berlin, ORZ
1086 Berlin
PSF 1297

Verzeichnis der Autoren

Aden,H. 3-1
Ahrens,K. 5-1
Altmeyer,K. 4-36
Bauer,G. 4-1
Bell,J.-P. 6-1
Bergholz,G. 6-9
Buchholz,B. 3-5
Burghardt,F. 6-18
Conrad,E. 2-1
Chmiel,G. 4-7
Clauß,M. 5-8
Detering,R. 4-8
Döring,P. 4-22
Dörre,W. 1-1
Falek,M. 1-17
Fellin,A. 1-116
Fischer,J. 4-26
Fischer,G. 5-8
Fuchs-Kittowski,K. o-1
Friedrich,S. 2-15
Frolow,A.K. 6-58
Gellermann,W.-D. 5-13
Gerhardt,W. o-16
Gernert,H. 2-25
Grabowski,J. 5-13; 5-22
Grafik,W. 6-1
Grebenikow,E.A. o-28
Gugel,G. 4-31
Hartwig,M. 5-31
Hartwig,W.-H. 2-35
Heckert,R. 6-24
Heiner,M. 6-36
Henkel,R. 4-32
Herrlich,O. 4-8;4-36
Höllering,H.-H. 3-12
Hopfer,R. 5-32
Horn,E. o-29,4-87
Horváth,J. 3-44
Hutschenreiter,J. 5-33
Jacob,H.-J. 4-37
Jakimenko,A.W. 1-94
Kade,G. 2-1
Kawka,R. o-27
Kerner,I.O. o-58;o-59
Kiosa,M.N. 1-94
Kiss,G. 4-45
Koitz,R. 1-23,1-95
Köhler,K. 6-31
König,H. 6-36
Koeziolowicz,R. 5-34
Kozma,L. 3-44
Krien,R. 3-16
Kronrádova 1-32
Kroß,G. o-60
Kühle,F. 4-8
Kühne,T. 2-42
Lassig,A.T. 4-22
Lemke,G. 2-46
Lorenz,M. 4-8
Mager,K. 5-39
Mätzel,K. 2-51
Maye,H.-W. 1-52,1-62
Meuche,H.-F. 1-71
Mikulecký,P. o-75
Miniberger,B. 1-32
Mollenhauer,E. 3-17

Morosow,A.W. o-76
Mühlenberg,E. 1-78
Müller,D. 3-81
Müller,K. 6-47
Müller,P. 6-1
Mysiak,A. 1-84

Neef,G. 2-54

Pälecke,B. 3-24
Piens,C. 6-47
Polák,J. 1-32
Porkert,K. 3-25
Prescher,G. 3-35
Protzner,S. 4-8

Racz,E. 3-44
Rasumowa,G.S. 6-58
Repin,W.M. 6-54
Rieck,K.-D. 6-55
Riewendt,P. o-47
Ringhand,K. 2-52
Rudeck,C. 1-95
Rüdiger,P. 5-47

Samarin,M.K. 3-54
Schiemangk,H. o-77
Schilling,A. 2-53,5-61
Schober,K. 3-12
Schönefeld,R. 4-53
Schreiber,E. 6-59
Schrewe,J. 4-53
Schröder,H. 2-1
Schubert,W. 4-36
Schuster,H. 4-66
Seidel,H. 2-1
Shelesnow,N.P. 1-94,6-58

Simon,U. 2-54
Sommerfeld,M. 6-78
Staudte,R. 5-62
Stein,E. 5-31
Stojković,V. 5-66
Strobel,R. 5-31
Strohmeyer,S. 4-77
Stuchlik,F. 6-6o

Trautewig,M. 1-95
Trautloft,R. 3-55
Tschirschwitz,R. 1-78

Wagner,B. 4-77
Wagner,K. 1-1o6,2-59
Waldmann,L. 3-68
Wegel,G. 4-81
Wenzlaff,B. o-1
Wenzlaff,R. 1-116
Werner,K. 3-73
Wiederänders,K. 5-67
Winkler,F. 4-87
Winke,H. 6-86
Winzerling,W. 6-6o

Zilahy,J.-P. 4-97
Zimmerling,R. 3-81
Zschockelt,P. 3-88